



Sun Java System Message Queue 3.7 UR1 Administration Guide



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

Part No: 819-4467-10
January 2007

Copyright 2006 Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95054 U.S.A. All rights reserved.

Sun Microsystems, Inc. has intellectual property rights relating to technology embodied in the product that is described in this document. In particular, and without limitation, these intellectual property rights may include one or more U.S. patents or pending patent applications in the U.S. and in other countries.

U.S. Government Rights – Commercial software. Government users are subject to the Sun Microsystems, Inc. standard license agreement and applicable provisions of the FAR and its supplements.

This distribution may include materials developed by third parties.

Parts of the product may be derived from Berkeley BSD systems, licensed from the University of California. UNIX is a registered trademark in the U.S. and other countries, exclusively licensed through X/Open Company, Ltd.

Sun, Sun Microsystems, the Sun logo, the Solaris logo, the Java Coffee Cup logo, docs.sun.com, Java, and Solaris are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

The OPEN LOOK and SunTM Graphical User Interface was developed by Sun Microsystems, Inc. for its users and licensees. Sun acknowledges the pioneering efforts of Xerox in researching and developing the concept of visual or graphical user interfaces for the computer industry. Sun holds a non-exclusive license from Xerox to the Xerox Graphical User Interface, which license also covers Sun's licensees who implement OPEN LOOK GUIs and otherwise comply with Sun's written license agreements.

Products covered by and information contained in this publication are controlled by U.S. Export Control laws and may be subject to the export or import laws in other countries. Nuclear, missile, chemical or biological weapons or nuclear maritime end uses or end users, whether direct or indirect, are strictly prohibited. Export or reexport to countries subject to U.S. embargo or to entities identified on U.S. export exclusion lists, including, but not limited to, the denied persons and specially designated nationals lists is strictly prohibited.

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

Copyright 2006 Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95054 U.S.A. Tous droits réservés.

Sun Microsystems, Inc. détient les droits de propriété intellectuelle relatifs à la technologie incorporée dans le produit qui est décrit dans ce document. En particulier, et ce sans limitation, ces droits de propriété intellectuelle peuvent inclure un ou plusieurs brevets américains ou des applications de brevet en attente aux Etats-Unis et dans d'autres pays.

Cette distribution peut comprendre des composants développés par des tierces personnes.

Certains composants de ce produit peuvent être dérivées du logiciel Berkeley BSD, licenciés par l'Université de Californie. UNIX est une marque déposée aux Etats-Unis et dans d'autres pays; elle est licenciée exclusivement par X/Open Company, Ltd.

Sun, Sun Microsystems, le logo Sun, le logo Solaris, le logo Java Coffee Cup, docs.sun.com, Java et Solaris sont des marques de fabrique ou des marques déposées de Sun Microsystems, Inc. aux Etats-Unis et dans d'autres pays. Toutes les marques SPARC sont utilisées sous licence et sont des marques de fabrique ou des marques déposées de SPARC International, Inc. aux Etats-Unis et dans d'autres pays. Les produits portant les marques SPARC sont basés sur une architecture développée par Sun Microsystems, Inc.

L'interface d'utilisation graphique OPEN LOOK et Sun a été développée par Sun Microsystems, Inc. pour ses utilisateurs et licenciés. Sun reconnaît les efforts de pionniers de Xerox pour la recherche et le développement du concept des interfaces d'utilisation visuelle ou graphique pour l'industrie de l'informatique. Sun détient une licence non exclusive de Xerox sur l'interface d'utilisation graphique Xerox, cette licence couvrant également les licenciés de Sun qui mettent en place l'interface d'utilisation graphique OPEN LOOK et qui, en outre, se conforment aux licences écrites de Sun.

Les produits qui font l'objet de cette publication et les informations qu'il contient sont régis par la législation américaine en matière de contrôle des exportations et peuvent être soumis au droit d'autres pays dans le domaine des exportations et importations. Les utilisations finales, ou utilisateurs finaux, pour des armes nucléaires, des missiles, des armes chimiques ou biologiques ou pour le nucléaire maritime, directement ou indirectement, sont strictement interdites. Les exportations ou réexportations vers des pays sous embargo des Etats-Unis, ou vers des entités figurant sur les listes d'exclusion d'exportation américaines, y compris, mais de manière non exclusive, la liste de personnes qui font objet d'un ordre de ne pas participer, d'une façon directe ou indirecte, aux exportations des produits ou des services qui sont régis par la législation américaine en matière de contrôle des exportations et la liste de ressortissants spécifiquement désignés, sont rigoureusement interdites.

LA DOCUMENTATION EST FOURNIE "EN L'ETAT" ET TOUTES AUTRES CONDITIONS, DECLARATIONS ET GARANTIES EXPRESSES OU TACITES SONT FORMELLEMENT EXCLUES, DANS LA MESURE AUTORISEE PAR LA LOI APPLICABLE, Y COMPRIS NOTAMMENT TOUTE GARANTIE IMPLICITE RELATIVE A LA QUALITE MARCHANDE, A L'APTITUDE A UNE UTILISATION PARTICULIERE OU A L'ABSENCE DE CONTREFACON.

Contents

Preface	21
 Part I Introduction to Message Queue Administration	 31
 1 Administrative Tasks and Tools	 33
Administrative Tasks	33
Administration in a Development Environment	33
Administration in a Production Environment	34
Administration Tools	36
Command Line Utilities	36
Administration Console	36
 2 Quick-Start Tutorial	 39
Starting the Administration Console	40
Administration Console Online Help	42
Working With Brokers	43
Starting a Broker	43
Adding a Broker to the Administration Console	43
▼ To Add a Broker to the Administration Console	43
Connecting to a Broker	45
▼ To Connect to a Broker	45
Viewing Connection Services	46
▼ To View Available Connection Services	46
Working With Physical Destinations	48
Creating a Physical Destination	48
▼ To Add a Physical Destination to a Broker	48
Viewing Physical Destination Properties	49

▼ To View or Modify the Properties of a Physical Destination	50
Purging Messages From a Physical Destination	52
▼ To Purge Messages From a Physical Destination	52
Deleting a Physical Destination	52
▼ To Delete a Physical Destination	53
Working With Object Stores	53
Adding an Object Store	53
▼ To Add an Object Store to the Administration Console	54
Connecting to an Object Store	55
▼ To Connect to an Object Store	56
Working With Administered Objects	56
Adding a Connection Factory	56
▼ To Add a Connection Factory to an Object Store	56
Adding a Destination	58
▼ To Add a Destination to an Object Store	58
Viewing Administered Object Properties	60
▼ To View or Modify the Properties of an Administered Object	60
Deleting an Administered Object	61
▼ To Delete an Administered Object	61
Running the Sample Application	61
▼ To Run the Sample Application	62
 Part II Administrative Tasks	 65
 3 Starting Brokers and Clients	 67
Preparing System Resources	67
Synchronizing System Clocks	67
Setting the File Descriptor Limit	68
Starting Brokers	68
Starting Brokers Interactively	68
Starting Brokers Automatically	69
Removing Brokers	72
Removing a Broker on Solaris or Linux	72
Removing a Windows Broker Service	73
Starting Clients	73

4	Configuring a Broker	75
	Broker Services	75
	Connection Services	76
	Routing Services	78
	Persistence Services	79
	Security Services	82
	Monitoring Services	85
	Setting Broker Properties	88
	Configuration Files	88
	Configuring a Persistent Data Store	91
	Configuring a File-Based Store	91
	Configuring a JDBC-Based Store	91
	▼ To Configure a JDBC-Based Data Store	92
	Securing Persistent Data	93
5	Managing a Broker	95
	Prerequisites	96
	Using the imqcmd Utility	96
	Displaying Help	96
	Displaying the Product Version	97
	Specifying the User Name and Password	97
	Specifying the Broker Name and Port	97
	Examples	98
	Displaying Broker Information	98
	Updating Broker Properties	99
	Pausing and Resuming a Broker	100
	Pausing a Broker	100
	Resuming a Broker	101
	Shutting Down and Restarting a Broker	101
	Displaying Broker Metrics	102
	Managing Connection Services	103
	Listing Connection Services	103
	Displaying Connection Service Information	104
	Updating Connection Service Properties	105
	Displaying Connection Service Metrics	105

Pausing and Resuming a Connection Service	106
Getting Information About Connections	107
Managing Durable Subscriptions	108
Managing Transactions	110
6 Managing Physical Destinations	113
Using the Command Utility	114
Subcommands	114
Creating a Physical Destination	115
Listing Physical Destinations	116
Displaying Information about Physical Destinations	117
Updating Physical Destination Properties	118
Pausing and Resuming Physical Destinations	119
Purging Physical Destinations	120
Destroying Physical Destinations	120
Compacting Physical Destinations	121
Monitoring a Physical Destination's Disk Utilization	121
Using the Dead Message Queue	123
Configuring Use of the Dead Message Queue	123
Managing the Dead Message Queue	124
Enabling Dead Message Logging	125
7 Managing Security	127
User Authentication	127
Using a Flat-File User Repository	128
Using an LDAP Server for a User Repository	133
User Authorization: The Access Control Properties File	135
Creating an Access Control Properties File	136
Syntax of Access Rules	136
How Permissions are Computed	137
Access Control for Connection Services	138
Access Control for Physical Destinations	139
Access Control for Auto-Created Physical Destinations	140
Message Encryption	141
Using Self-Signed Certificates	141

▼ To Set Up an SSL-Based Connection Service Using Self-Signed Certificates	142
Using Signed Certificates	146
▼ To Use a Signed Certificate	147
Password Files	149
Security Concerns	150
Password File Contents	150
Connecting Through a Firewall	151
▼ To Enable Broker Connections Through a Firewall	152
Audit Logging	152
8 Managing Administered Objects	153
Object Stores	153
LDAP Server Object Stores	153
File-System Object Stores	155
Administered Object Attributes	156
Connection Factory Attributes	156
Destination Attributes	162
Using the Object Manager Utility	163
Adding Administered Objects	164
Deleting Administered Objects	165
Listing Administered Objects	166
Viewing Administered Object Information	166
Modifying Administered Object Attributes	167
Using Command Files	167
9 Working With Broker Clusters	171
Cluster Configuration Properties	171
Setting Cluster Properties for Individual Brokers	172
Using a Cluster Configuration File	172
Managing Clusters	173
Connecting Brokers	173
Adding Brokers to a Cluster	174
▼ To Add a New Broker to a Cluster Using a Cluster Configuration File	174
Removing Brokers From a Cluster	175
Master Broker	176

Managing the Configuration Change Record	176
▼ To Restore the Configuration Change Record	177
When a Master Broker Is Unavailable	177
10 Monitoring a Broker	179
Introduction to Monitoring Tools	179
Configuring and Using Broker Logging	181
Default Logging Configuration	181
Log Message Format	182
Changing the Logger Configuration	182
▼ To Change the Logger Configuration for a Broker	182
Displaying Metrics Interactively	186
imqcmd metrics	186
Using the metrics Subcommand to Display Metrics Data	188
▼ To Use the metrics Subcommand	188
Metrics Outputs: imqcmd metrics	188
imqcmd query	189
Writing an Application to Monitor Brokers	190
Setting Up Message-Based Monitoring	191
▼ To Set Up Message-based Monitoring	191
Security and Access Considerations	192
Metrics Outputs: Metrics Messages	193
11 Analyzing and Tuning a Message Service	195
About Performance	195
The Performance Tuning Process	195
Aspects of Performance	196
Benchmarks	196
Baseline Use Patterns	197
Factors Affecting Performance	198
▼ Message Delivery Steps	199
Application Design Factors Affecting Performance	200
Message Service Factors Affecting Performance	204
Adjusting Configuration To Improve Performance	208
System Adjustments	208

Broker Adjustments	211
Client Runtime Message Flow Adjustments	213
12 Troubleshooting Problems	217
A Client Cannot Establish a Connection	217
Connection Throughput Is Too Slow	221
A Client Cannot Create a Message Producer	222
Message Production Is Delayed or Slowed	223
Messages Are Backlogged	226
Broker Throughput Is Sporadic	230
Messages Are Not Reaching Consumers	231
Dead Message Queue Contains Messages	234
Part III Reference	241
13 Command Line Reference	243
Command Line Syntax	243
Broker Utility	244
Command Utility	248
Broker Management	250
Connection Service Management	251
Connection Management	252
Physical Destination Management	252
Durable Subscription Management	254
Transaction Management	255
General Command Utility Options	255
Object Manager Utility	256
Database Manager Utility	257
User Manager Utility	259
Service Administrator Utility	260
Key Tool Utility	261
14 Broker Properties Reference	263
Connection Properties	263

Routing Properties	265
Persistence Properties	270
File-Based Persistence	270
JDBC-Based Persistence	271
Security Properties	274
Monitoring Properties	278
Cluster Configuration Properties	282
Alphabetical List of Broker Properties	283
15 Physical Destination Property Reference	289
Physical Destination Properties	289
16 Administered Object Attribute Reference	293
Connection Factory Attributes	293
Connection Handling	293
Client Identification	297
Reliability and Flow Control	297
Queue Browser and Server Sessions	298
Standard Message Properties	299
Message Header Overrides	300
Destination Attributes	300
SOAP Endpoint Attributes	301
17 JMS Resource Adapter Property Reference	303
ResourceAdapter JavaBean	303
ManagedConnectionFactory JavaBean	305
ActivationSpec JavaBean	306
18 Metrics Reference	309
JVM Metrics	309
Brokerwide Metrics	310
Connection Service Metrics	312
Destination Metrics	313

Part IV	Appendixes	319
A	Platform-Specific Locations of Message Queue Data	321
	Solaris	321
	Linux	322
	Windows	323
B	Stability of Message Queue Interfaces	325
	Classification Scheme	325
	Interface Stability	326
C	HTTP/HTTPS Support	329
	HTTP/HTTPS Support Architecture	329
	Enabling HTTP Support	331
	▼ To Enable HTTP Support	331
	Step 1. Deploy the HTTP Tunnel Servlet	331
	Step 2. Configure the httpjms Connection Service	334
	▼ To Activate the httpjms Connection Service	334
	Step 3. Configure an HTTP Connection	336
	Enabling HTTPS Support	337
	▼ To Enable HTTPS Support	338
	Step 1. Generating a Self-signed Certificate for the HTTPS Tunnel Servlet	338
	Step 2. Modifying the HTTP Tunnel Servlet .war File's Descriptor File	339
	▼ To Modify the HTTPS Tunnel Servlet .war File	339
	Step 3. Deploying the HTTPS Tunnel Servlet	340
	Step 4. Configuring the httpsjms Connection Service	343
	▼ To Activate the httpsjms Connection Service	344
	Step 5. Configuring an HTTPS Connection	345
	Troubleshooting	348
	Server or Broker Failure	348
	Client Failure to Connect Through the Tunnel Servlet	348
	▼ If a Client Cannot Connect	348

- D Frequently Used Command Utility Commands 349**
 - Syntax 349
 - Broker and Cluster Management 349
 - Broker Configuration Properties (-o option) 350
 - Service and Connection Management 350
 - Durable Subscriber Management 351
 - Transaction Management 351
 - Destination Management 351
 - Destination Configuration Properties (-o option) 351
 - Metrics 352

- Index 353**

Figures

FIGURE 1-1	Local and Remote Administration Utilities	37
FIGURE 2-1	Administration Console Window	41
FIGURE 2-2	Administration Console Help Window	42
FIGURE 2-3	Add Broker Dialog Box	44
FIGURE 2-4	Broker Displayed in Administration Console Window	45
FIGURE 2-5	Connect to Broker Dialog Box	46
FIGURE 2-6	Viewing Connection Services	47
FIGURE 2-7	Service Properties Dialog Box	47
FIGURE 2-8	Add Broker Destination Dialog Box	49
FIGURE 2-9	Broker Destination Properties Dialog Box	51
FIGURE 2-10	Durable Subscriptions Panel	52
FIGURE 2-11	Add Object Store Dialog Box	54
FIGURE 2-12	Object Store Displayed in Administration Console Window	55
FIGURE 2-13	Add Connection Factory Object Dialog Box	57
FIGURE 2-14	Add Destination Object Dialog Box	59
FIGURE 2-15	Destination Object Displayed in Administration Console Window	60
FIGURE 4-1	Persistent Data Storage	80
FIGURE 4-2	Security Support	83
FIGURE 4-3	Monitoring Support	86
FIGURE 4-4	Broker Configuration Files	89
FIGURE 11-1	Message Delivery Through a Message Queue Service	199
FIGURE 11-2	Transport Protocol Speeds	206
FIGURE C-1	HTTP/HTTPS Support Architecture	330

Tables

TABLE 4-1	Message Queue Connection Services	76
TABLE 4-2	Metric Topic Destinations	87
TABLE 5-1	Message Queue Connection Services	103
TABLE 5-2	Connection Service Properties Updated by <code>imqcmd</code>	105
TABLE 6-1	Physical Destination Subcommands for the Command Utility	114
TABLE 6-2	Physical Destination Disk Utilization Metrics	122
TABLE 6-3	Dead Message Queue Treatment of Standard Physical Destination Properties	124
TABLE 7-1	Initial Entries in User Repository	128
TABLE 7-2	<code>imqusermgr</code> Options	130
TABLE 7-3	Syntactic Elements of Access Rules	137
TABLE 7-4	Elements of Physical Destination Access Control Rules	140
TABLE 7-5	Distinguished Name Information Required for a Self-Signed Certificate	143
TABLE 7-6	Commands That Use Passwords	150
TABLE 7-7	Passwords in a Password File	151
TABLE 7-8	Broker Configuration Properties for Static Port Addresses	151
TABLE 8-1	LDAP Object Store Attributes	154
TABLE 8-2	File-system Object Store Attributes	155
TABLE 10-1	Benefits and Limitations of Metrics Monitoring Tools	180
TABLE 10-2	Logging Levels	182
TABLE 10-3	<code>imqcmd metrics</code> Subcommand Syntax	186
TABLE 10-4	<code>imqcmd metrics</code> Subcommand Options	187
TABLE 10-5	<code>imqcmd query</code> Subcommand Syntax	190
TABLE 10-6	Metric Topic Destinations	190
TABLE 11-1	Comparison of High-Reliability and High-Performance Scenarios	201
TABLE 13-1	Broker Utility Options	244
TABLE 13-2	Command Utility Subcommands for Broker Management	250
TABLE 13-3	Command Utility Subcommands for Connection Service Management	251
TABLE 13-4	Command Utility Subcommands for Connection Service Management	252

TABLE 13-5	Command Utility Subcommands for Physical Destination Management	253
TABLE 13-6	Command Utility Subcommands for Durable Subscription Management	254
TABLE 13-7	Command Utility Subcommands for Transaction Management	255
TABLE 13-8	General Command Utility Options	255
TABLE 13-9	Object Manager Subcommands	256
TABLE 13-10	Object Manager Options	256
TABLE 13-11	Database Manager Subcommands	258
TABLE 13-12	Database Manager Options	258
TABLE 13-13	User Manager Subcommands	259
TABLE 13-14	General User Manager Options	260
TABLE 13-15	Service Administrator Subcommands	260
TABLE 13-16	Service Administrator Options	261
TABLE 14-1	Broker Connection Properties	263
TABLE 14-2	Broker Routing Properties	265
TABLE 14-3	Broker Properties for Auto-Created Destinations	267
TABLE 14-4	Global Broker Persistence Property	270
TABLE 14-5	Broker Properties for File-Based Persistence	270
TABLE 14-6	Broker Properties for JDBC-Based Persistence	272
TABLE 14-7	Broker Security Properties	274
TABLE 14-8	Broker Monitoring Properties	278
TABLE 14-9	Broker Properties for Cluster Configuration	282
TABLE 14-10	Alphabetical List of Broker Properties	284
TABLE 15-1	Physical Destination Properties	289
TABLE 16-1	Connection Factory Attributes for Connection Handling	294
TABLE 16-2	Message Broker Addressing Schemes	295
TABLE 16-3	Message Broker Address Examples	296
TABLE 16-4	Connection Factory Attributes for Client Identification	297
TABLE 16-5	Connection Factory Attributes for Reliability and Flow Control	297
TABLE 16-6	Connection Factory Attributes for Queue Browser and Server Sessions	299
TABLE 16-7	Connection Factory Attributes for Standard Message Properties	299
TABLE 16-8	Connection Factory Attributes for Message Header Overrides	300
TABLE 16-9	Destination Attributes	301
TABLE 16-10	SOAP Endpoint Attributes	301
TABLE 17-1	Resource Adapter Properties	304
TABLE 17-2	Managed Connection Factory Properties	305
TABLE 17-3	Activation Specification Properties	306

TABLE 18-1	JVM Metrics	309
TABLE 18-2	Brokerwide Metrics	310
TABLE 18-3	Connection Service Metrics	312
TABLE 18-4	Destination Metrics	313
TABLE A-1	Message Queue Data Locations on Solaris Platform	321
TABLE A-2	Message Queue Data Locations on Linux Platform	322
TABLE A-3	Message Queue Data Locations on Windows Platform	323
TABLE B-1	Interface Stability Classification Scheme	325
TABLE B-2	Stability of Message Queue Interfaces	326
TABLE D-1	Broker Configuration Properties (-o option)	350
TABLE D-2	Destination Configuration Properties (-o option)	351

Examples

EXAMPLE 3-1	Displaying Broker Service Startup Options	72
EXAMPLE 8-1	Adding a Connection Factory	164
EXAMPLE 8-2	Adding a Destination to an LDAP Object Store	165
EXAMPLE 8-3	Adding a Destination to a File-System Object Store	165
EXAMPLE 8-4	Deleting an Administered Object	166
EXAMPLE 8-5	Listing All Administered Objects	166
EXAMPLE 8-6	Listing Administered Objects of a Specific Type	166
EXAMPLE 8-7	Viewing Administered Object Information	167
EXAMPLE 8-8	Modifying an Administered Object's Attributes	167
EXAMPLE 8-9	Object Manager Command File Syntax	168
EXAMPLE 8-10	Example Command File	168
EXAMPLE 8-11	Partial Command File	169
EXAMPLE 8-12	Using a Partial Command File	169

Preface

This Sun Java System™ *Message Queue Administration Guide* provides background and information needed by system administrators to set up and manage a Sun Java System Message Queue™ messaging system.

Who Should Use This Book

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

Before You Read This Book

Before reading this manual, you should read the *Sun Java System Message Queue 3.7 URI Technical Overview* to become familiar with Message Queue’s implementation of the Java Message Service specification, with the components of the Message Queue service, and with the basic process of developing, deploying, and administering a Message Queue application.

How This Book Is Organized

[Table P-1](#) describes the contents of this manual.

TABLE P-1 Contents of This Manual

Part/Chapter	Description
Part I, Introduction to Message Queue Administration	
Chapter 1, Administrative Tasks and Tools	Introduces Message Queue administrative tasks and tools.

TABLE P-1 Contents of This Manual (Continued)

Part/Chapter	Description
Chapter 2, Quick-Start Tutorial	Provides a hands-on tutorial to acquaint you with the Message Queue Administration Console.
Part II, Administrative 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.
Chapter 5, Managing a Broker	Describes broker management tasks.
Chapter 6, Managing Physical Destinations	Describes management tasks relating to physical destinations.
Chapter 7, Managing Security	Describes security-related tasks, such as managing password files, authentication, authorization, and encryption.
Chapter 8, Managing Administered Objects	Describes the object store and shows how to perform tasks related to administered objects (connection factories and destinations).
Chapter 9, Working With Broker Clusters	Describes how to set up and manage a cluster of Message Queue brokers.
Chapter 10, Monitoring a Broker	Describes how to set up and use Message Queue monitoring facilities.
Chapter 11, Analyzing and Tuning a Message Service	Describes techniques for analyzing and optimizing message service performance.
Chapter 12, Troubleshooting Problems	Provides suggestions for determining the cause of common Message Queue problems and the actions you can take to resolve them.
Part III, Reference	
Chapter 13, Command Line Reference	Provides syntax and descriptions for Message Queue command line utilities.
Chapter 14, Broker Properties Reference	Describes the configuration properties of Message Queue message brokers.
Chapter 15, Physical Destination Property Reference	Describes the configuration properties of physical destinations.
Chapter 16, Administered Object Attribute Reference	Describes the configuration properties of administered objects (connection factories and destinations).
Chapter 17, JMS Resource Adapter Property Reference	Describes the configuration properties of the Message Queue Resource Adapter for use with an application server.

TABLE P-1 Contents of This Manual (Continued)

Part/Chapter	Description
Chapter 18, Metrics Reference	Describes the metric information that a Message Queue message broker can provide for monitoring, turning, and diagnostic purposes. .
Part IV, Appendixes	
Appendix A, Platform-Specific Locations of Message Queue Data	Lists the locations of Message Queue files on each supported platform.
Appendix B, Stability of Message Queue Interfaces	Describes the stability of various Message Queue interfaces.
Appendix C, HTTP/HTTPS Support	Describes how to set up and use the Hypertext Transfer Protocol (HTTP) for Message Queue communication.
Appendix D, Frequently Used Command Utility Commands	Lists some frequently used Message Queue Command utility (<code>imqcmd</code>) commands.

Related Documentation

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

TABLE P-2 Message Queue Documentation Set

Document	Audience	Description
<i>Sun Java System Message Queue 3.7 UR1 Installation Guide</i>	Developers and administrators	Explains how to install Message Queue software on Solaris, Linux, and Windows platforms.
<i>Sun Java System Message Queue 3.7 UR1 Release Notes</i>	Developers and administrators	Includes descriptions of new features, limitations, and known bugs, as well as technical notes.
<i>Sun Java System Message Queue 3.7 UR1 Technical Overview</i>	Developers and administrators	Describes Message Queue concepts, features, and components.
<i>Sun Java System Message Queue 3.7 UR1 Developer's Guide for Java Clients</i>	Developers	Provides a quick-start tutorial and programming information for developers of Java client programs using the Message Queue implementation of the JMS or SOAP/JAXM APIs.
<i>Sun Java System Message Queue 3.7 UR1 Administration Guide</i>	Administrators, also recommended for developers	Provides background and information needed to perform administration tasks using Message Queue administration tools.

TABLE P-2 Message Queue Documentation Set (Continued)

Document	Audience	Description
<i>Sun Java System Message Queue 3.7 UR1 Developer's Guide for C Clients</i>	Developers	Provides programming and reference documentation for developers of C client programs using the Message Queue C implementation of the JMS API (C-API).

Online Help

Message Queue 3.7 UR1 includes command-line utilities for performing Message Queue message service administration tasks.

Message Queue 3.7 UR1 also includes a graphical user interface (GUI) administration tool, the Administration Console (imqadmin). Context-sensitive help is included in the Administration Console; see [“Administration Console Online Help” on page 42](#).

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. It includes standard JMS API documentation, as well as Message Queue-specific APIs for Message Queue administered objects, which are of value to developers of messaging applications.

Example Client Applications

Message Queue provides a number of example client applications to assist developers.

Example Java Client Applications

Example Java client applications are located in the following directories, depending on platform. See the README file located in these directories and in each of their subdirectories.

Platform	Location
Solaris	/usr/demo/imq/
Linux	/opt/sun/mq/examples
Windows	IMQ_HOME/demo/

Example C Client Programs

Example C client applications are located in the following directories, depending on platform. See the README file located in these directories.

Platform	Location
Solaris	/opt/SUNWimq/demo/C/
Linux	/opt/sun/mq/examples/C/
Windows	IMQ_HOME/demo/C/

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.

Directory Variable Conventions

Message Queue makes use of three directory variables; how they are set varies from platform to platform. [Table P-3](#) describes these variables and how they are used on the Solaris, Linux, and Windows platforms.

Note – The information in [Table P-3](#) applies only to the standalone installation of Message Queue. When Message Queue is installed and run as part of an Application Server installation, the values of the directory variables are set differently: IMQ_HOME is set to *appServer_install_dir/imq* (where *appServer_install_dir* is the Application Server installation directory), and IMQ_VARHOME to *appServer_domainName_dir/imq* (where *appServer_domainName_dir* is the domain directory for the domain starting the Message Queue broker).

TABLE P-3 Directory Variable Conventions

Variable	Description
IMQ_HOME	<p>Used in Message Queue documentation to refer to the Message Queue base directory (root installation directory):</p> <ul style="list-style-type: none">■ 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 in Solaris and Linux.■ On Windows, the root Message Queue installation directory is set to the directory in which you unzip the Message Queue bundle.
IMQ_VARHOME	<p>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.</p> <ul style="list-style-type: none">■ 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 Linux, IMQ_VARHOME defaults to the /var/opt/sun/mq directory.■ On Windows, IMQ_VARHOME defaults to the IMQ_HOME/var directory.

TABLE P-3 Directory Variable Conventions (Continued)

Variable	Description
IMQ_JAVAHOME	<p>An environment variable that points to the location of the Java runtime (JRE) required by Message Queue executables:</p> <ul style="list-style-type: none"> 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. <code>/usr/jdk/entsys-j2se</code> <code>/usr/jdk/latest</code> <code>/usr/jdk/jdk1.5.*</code> <code>/usr/jdk/j2sdk1.5.*</code> <code>/usr/j2se</code> Solaris 10: <code>/usr/jdk/entsys-j2se</code> <code>/usr/jdk/java</code> <code>/usr/jdk/latest</code> <code>/usr/j2se</code> 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. <code>/usr/jdk/entsys-j2se</code> <code>/usr/java/jre1.5.*</code> <code>/usr/java/jdk1.5.*</code> <code>/usr/java/jre1.4.2*</code> <code>/usr/java/j2sdk1.4.2*</code> On Windows, IMQ_JAVAHOME will be set to point to an existing Java runtime if a supported version is found on the system. If a supported version is not found, one will be installed.

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 (/).

Typographic Conventions

The following table describes the typographic conventions that are used in this book.

TABLE P-4 Typographic Conventions

Typeface	Meaning	Example
AaBbCc123	The names of commands, files, and directories, and onscreen computer output	Edit your <code>.login</code> file. Use <code>ls -a</code> to list all files. <code>machine_name%</code> you have mail.
AaBbCc123	What you type, contrasted with onscreen computer output	<code>machine_name%</code> su Password:
<i>aabbcc123</i>	Placeholder: replace with a real name or value	The command to remove a file is <i>rm filename</i> .
<i>AaBbCc123</i>	Book titles, new terms, and terms to be emphasized	Read Chapter 6 in the <i>User's Guide</i> . <i>A cache</i> is a copy that is stored locally. Do <i>not</i> save the file. Note: Some emphasized items appear bold online.

Shell Prompts in Command Examples

The following table shows the default UNIX® system prompt and superuser prompt for the C shell, Bourne shell, Korn shell, and Windows operating system.

TABLE P-5 Shell Prompts

Shell	Prompt
C shell	<code>machine_name%</code>
C shell for superuser	<code>machine_name#</code>
Bourne shell and Korn shell	<code>\$</code>
Bourne shell and Korn shell for superuser	<code>#</code>
Windows	<code>C:\</code>

Symbol Conventions

The following table explains symbols that might be used in this book.

TABLE P-6 Symbol Conventions

Symbol	Description	Example	Meaning
[]	Contains optional arguments and command options.	<code>ls [-l]</code>	The <code>-l</code> option is not required.
{ }	Contains a set of choices for a required command option.	<code>-d {y n}</code>	The <code>-d</code> option requires that you use either the <code>y</code> argument or the <code>n</code> argument.
<code>\${ }</code>	Indicates a variable reference.	<code>\${com.sun.javaRoot}</code>	References the value of the <code>com.sun.javaRoot</code> variable.
-	Joins simultaneous multiple keystrokes.	Control-A	Press the Control key while you press the A key.
+	Joins consecutive multiple keystrokes.	Ctrl+A+N	Press the Control key, release it, and then press the subsequent keys.
→	Indicates menu item selection in a graphical user interface.	File → New → Templates	From the File menu, choose New. From the New submenu, choose Templates.

Documentation, Support, and Training

The Sun web site provides information about the following additional resources:

- Documentation (<http://www.sun.com/documentation/>)
- Support (<http://www.sun.com/support/>)
- Training (<http://www.sun.com/training/>)

Searching Sun Product Documentation

Besides searching Sun product documentation from the docs.sun.com web site, you can use a search engine by typing the following syntax in the search field:

```
search-term site:docs.sun.com
```

For example, to search for “broker,” type the following:

```
broker site:docs.sun.com
```

To include other Sun web sites in your search (for example, java.sun.com, www.sun.com, and developers.sun.com), use “sun.com” in place of “docs.sun.com” in the search field.

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 or alleged to be caused by or in connection with 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 full document title and part number. The part number is a 7-digit or 9-digit number that can be found on the book's title page or in the document's URL. For example, the part number of this book is 819-4467.



PART I

Introduction to Message Queue Administration

- [Chapter 1](#)
- [Chapter 2](#)

Administrative Tasks and Tools

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

- [“Administrative Tasks” on page 33](#)
- [“Administration Tools” on page 36](#)

Administrative Tasks

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

Administration in a Development Environment

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

- Simple startup of brokers for use in testing
- Administered objects instantiated in client code rather than created administratively
- Auto-created destinations
- File-system object store
- File-based persistence

- File-based user repository
- No master broker in multiple-broker clusters

Administration in a Production Environment

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

Setup Operations

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

Administrator security

- Setting the password for the default administrative user (`admin`) ([“Changing the Default Administrator Password” on page 132](#))
- Controlling individual or group access to the administrative connection service ([“Access Control for Connection Services” on page 138](#)) and the dead message queue ([“Access Control for Physical Destinations” on page 139](#))
- Regulating administrative group access to a file-based or Lightweight Directory Access Protocol (LDAP) user repository ([“Groups” on page 130](#), [“Setting Up Access Control for Administrators” on page 134](#))

General security

- Managing the contents of a file-based user repository ([“Populating and Managing a User Repository” on page 131](#)) or configuring the broker to use an existing LDAP user repository ([“Editing the Instance Configuration File” on page 133](#))
- Controlling the operations that individual users or groups are authorized to perform ([“User Authorization: The Access Control Properties File” on page 135](#))
- Setting up encryption services using the Secure Socket Layer (SSL) ([“Message Encryption” on page 141](#))

Administered objects

- Setting up and configuring an LDAP object store ([“LDAP Server Object Stores” on page 153](#))
- Creating connection factories and destinations ([“Adding Administered Objects” on page 164](#))

Broker clusters

- Creating a cluster configuration file ([“Using a Cluster Configuration File” on page 172](#))
- Designating a master broker ([“Master Broker” on page 176](#))

Persistence

- Configuring a broker to use a persistent store ([“Configuring a Persistent Data Store” on page 91](#)).

Memory management

- Setting a destination’s configuration properties to optimize its memory usage ([“Updating Physical Destination Properties” on page 118, Chapter 15](#))

Maintenance Operations

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

Broker administration and tuning

- Using broker metrics to tune and reconfigure a broker ([Chapter 11](#))
- Managing broker memory resources ([“Routing Services” on page 78](#))
- Creating and managing broker clusters to balance message load ([Chapter 9](#))
- Recovering failed brokers ([“Starting Brokers” on page 68](#)).

Administered objects

- Adjusting connection factory attributes to ensure the correct behavior of client applications ([“Connection Factory Attributes” on page 156](#))
- Monitoring and managing physical destinations ([Chapter 6](#))
- Controlling user access to destinations ([“Access Control for Physical Destinations” on page 139](#))

Client management

- Monitoring and managing durable subscriptions (see [“Managing Durable Subscriptions” on page 108](#)).
- Monitoring and managing transactions (see [“Managing Transactions” on page 110](#)).

Administration Tools

Message Queue administration tools fall into two categories:

- Command line utilities
- The graphical Administration Console

Command Line Utilities

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

- The *Broker utility* (`imqbrokerd`) starts up brokers and specifies their configuration properties, including connecting them together into a cluster.
- The *Command utility* (`imqcmd`) controls brokers and their resources and manages physical destinations.
- The *Object Manager utility* (`imqobjmgr`) manages provider-independent *administered objects* in an object store accessible via the Java Naming and Directory Interface (JNDI).
- The *Database Manager utility* (`imqdbmgr`) creates and manages databases for persistent storage that conform to the Java Database Connectivity (JDBC) standard.
- The *User Manager utility* (`imqusermgr`) populates a file-based user repository for user authentication and authorization.
- The *Service Administrator utility* (`imqsvcadm`) installs and manages a broker as a Windows service.
- The *Key Tool utility* (`imqkeytool`) generates self-signed certificates for Secure Socket Layer (SSL) authentication.

See [Chapter 13](#) for detailed information on the use of these utilities.

Administration Console

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

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

However, you cannot use the Administration Console to perform such tasks as starting up a broker, creating broker clusters, managing a JDBC database or a user repository, installing a broker as a Windows service, or generating SSL certificates. For these, you need the other

command line utilities (Broker, Database Manager, User Manager, Service Administrator, and Key Tool), which cannot operate remotely and must be run on the same host as the broker they manage (see [Figure 1-1](#)).

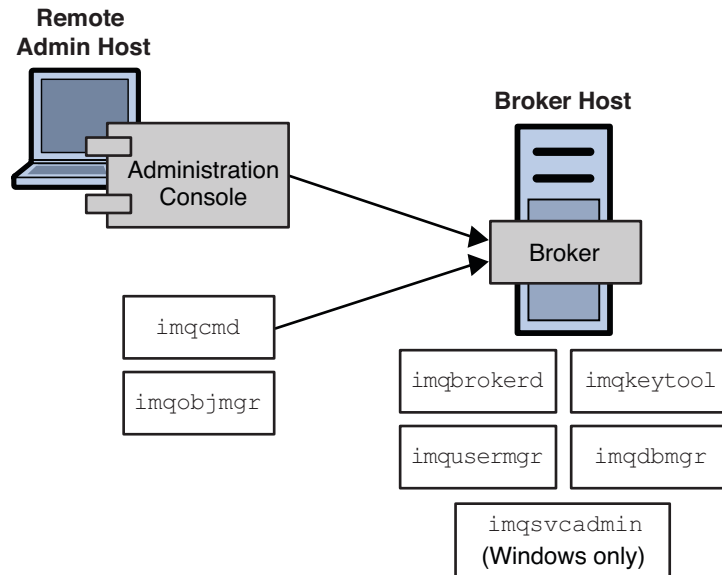


FIGURE 1-1 Local and Remote Administration Utilities

See [Chapter 2](#) for a brief, hands-on introduction to the Administration Console. More detailed information on its use is available through its own help facility.

Quick-Start Tutorial

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

- “Starting the Administration Console” on page 40
- “Administration Console Online Help” on page 42
- “Working With Brokers” on page 43
- “Working With Physical Destinations” on page 48
- “Working With Object Stores” on page 53
- “Working With Administered Objects” on page 56
- “Running the Sample Application” on page 61

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

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

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

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

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

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

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

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

Starting the Administration Console

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

- On Solaris, enter the command

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

- On Linux, enter the command

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

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

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

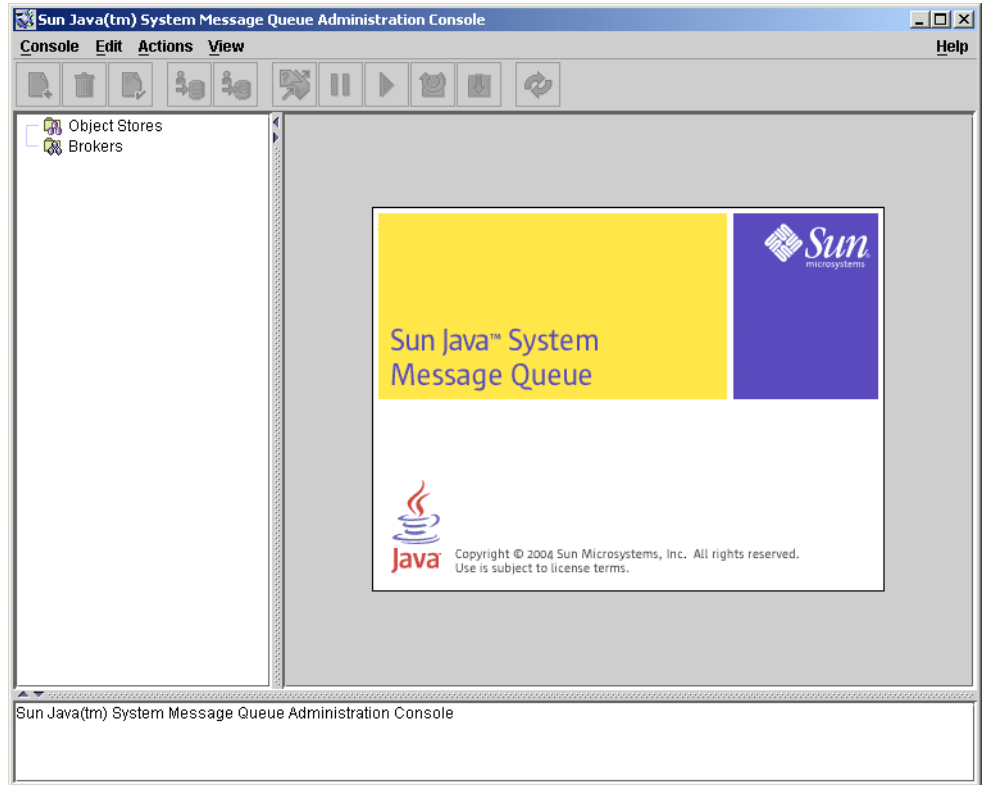


FIGURE 2-1 Administration Console Window

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

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

Administration Console Online Help

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

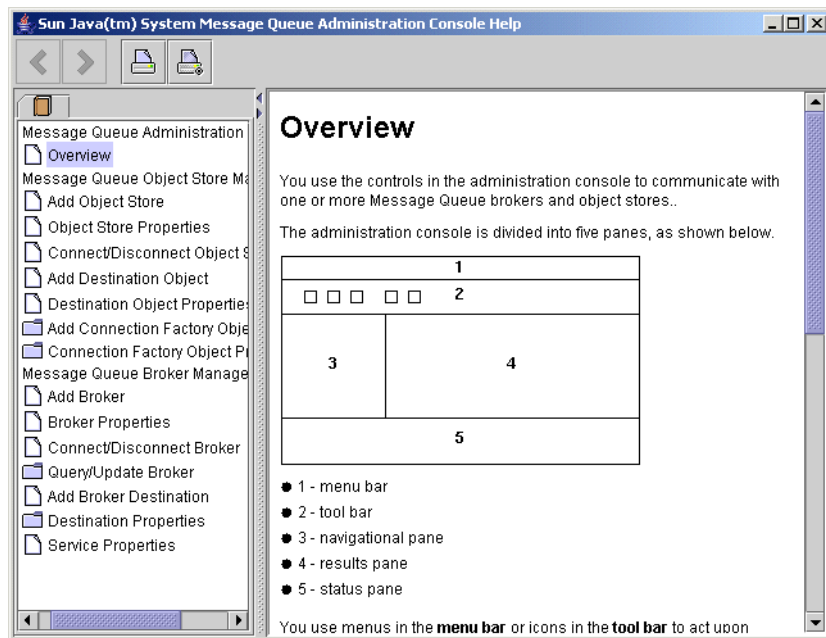


FIGURE 2-2 Administration Console Help Window

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

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

Working With Brokers

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

Starting a Broker

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

- On Solaris, enter this command:

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

- On Linux, enter this command:

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

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

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

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

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

Adding a Broker to the Administration Console

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

▼ To Add a Broker to the Administration Console

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

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

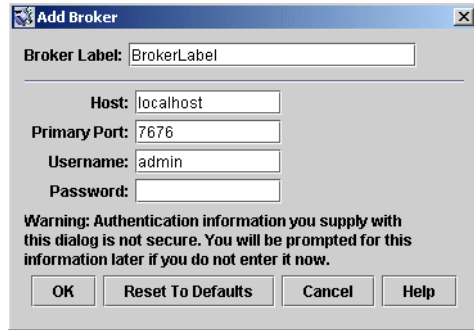


FIGURE 2-3 Add Broker Dialog Box

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

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

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

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

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

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

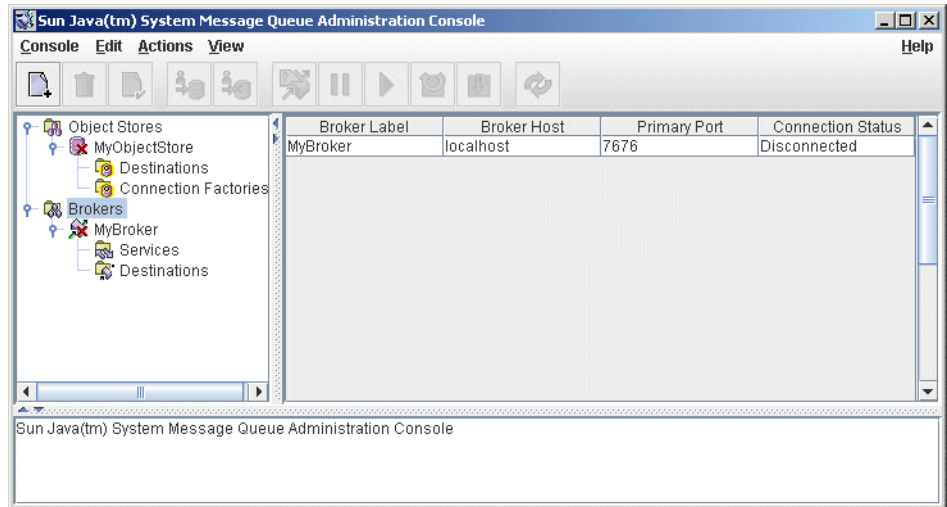


FIGURE 2-4 Broker Displayed in Administration Console Window

Once you have added a broker, you can use the Properties command on the Actions menu (or the pop-up context menu) to display a Broker Properties dialog box, similar to the Add Broker dialog shown in “[Adding a Broker to the Administration Console](#)” on page 43, to view or modify any of its properties.

Connecting to a Broker

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

▼ To Connect to a Broker

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

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

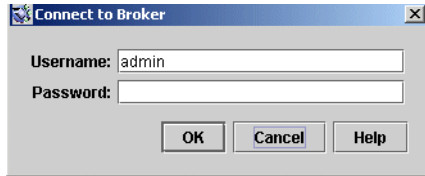


FIGURE 2-5 Connect to Broker Dialog Box

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

The dialog box initially displays the default user name, `admin`. In a real-world environment, you should establish secure user names and passwords as soon as possible (see [“User Authentication” on page 127](#)); for this exercise, simply use the default value.

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

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

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

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

Viewing Connection Services

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

▼ To View Available Connection Services

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

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

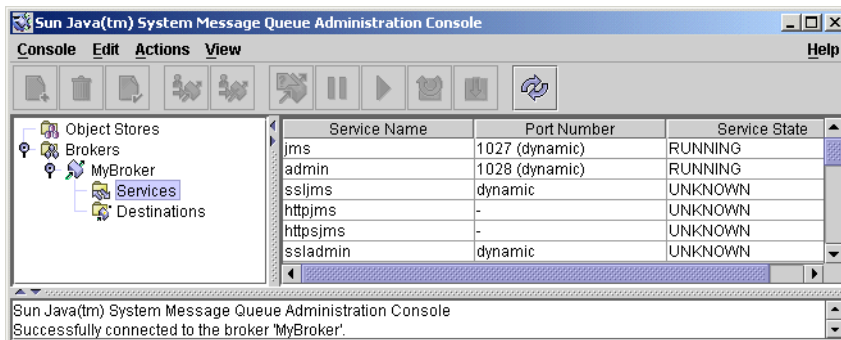


FIGURE 2-6 Viewing Connection Services

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

For this exercise, select the name `jms`.

- 3 **Choose Properties from the Actions menu.**

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

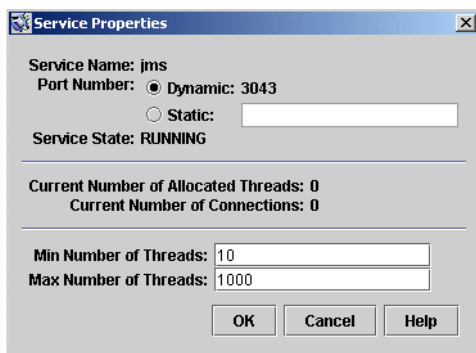


FIGURE 2-7 Service Properties Dialog Box

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

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

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

Working With Physical Destinations

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

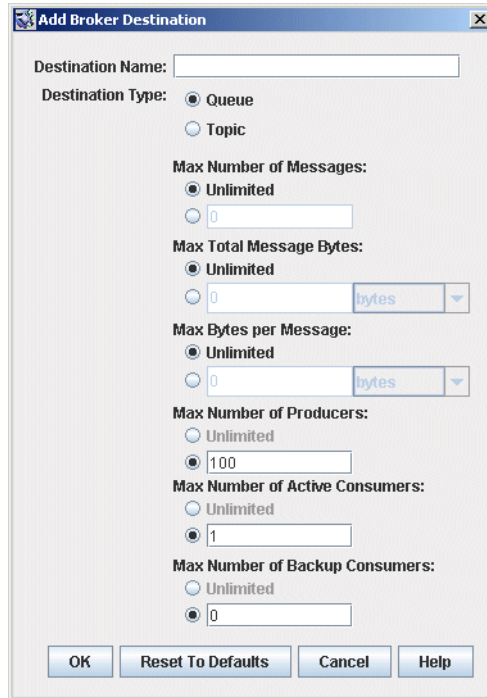
Creating a Physical Destination

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

▼ To Add a Physical Destination to a Broker

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

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



The dialog box is titled "Add Broker Destination". It contains the following fields and controls:

- Destination Name:** A text input field.
- Destination Type:** Two radio buttons: "Queue" (selected) and "Topic".
- Max Number of Messages:** Two radio buttons: "Unlimited" (selected) and "0".
- Max Total Message Bytes:** Two radio buttons: "Unlimited" (selected) and "0". A "bytes" label and a dropdown arrow are next to the "0" field.
- Max Bytes per Message:** Two radio buttons: "Unlimited" (selected) and "0". A "bytes" label and a dropdown arrow are next to the "0" field.
- Max Number of Producers:** Two radio buttons: "Unlimited" and "100" (selected).
- Max Number of Active Consumers:** Two radio buttons: "Unlimited" and "1" (selected).
- Max Number of Backup Consumers:** Two radio buttons: "Unlimited" and "0" (selected).

At the bottom are four buttons: "OK", "Reset To Defaults", "Cancel", and "Help".

FIGURE 2-8 Add Broker Destination Dialog Box

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

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

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

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

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

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

The new destination will appear in the result pane.

Viewing Physical Destination Properties

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

▼ To View or Modify the Properties of a Physical Destination

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

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

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

- 3 **Choose Properties from the Actions menu.**

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

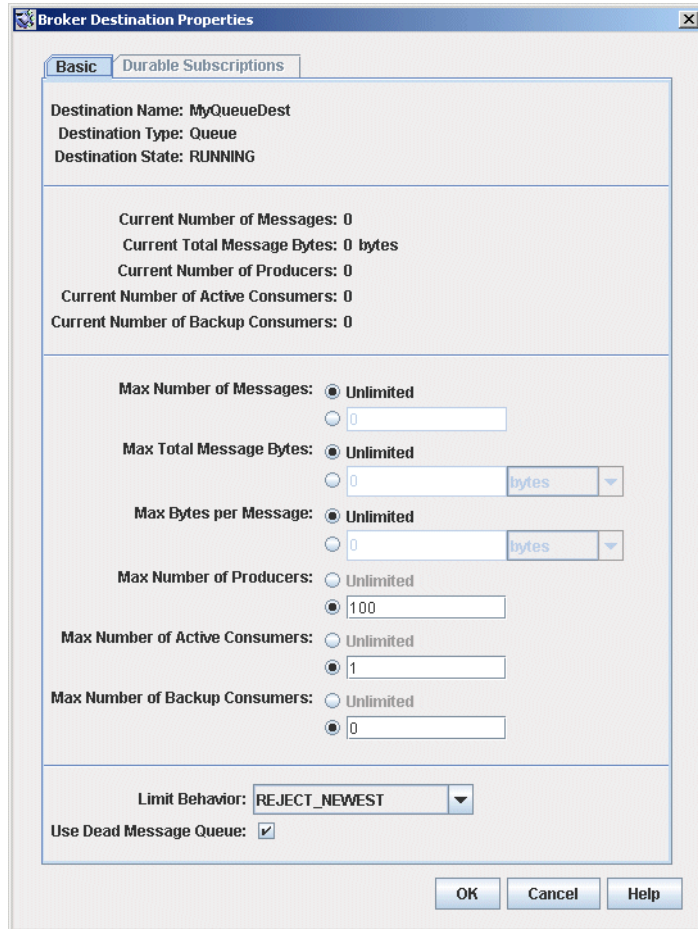


FIGURE 2-9 Broker Destination Properties Dialog Box

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

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

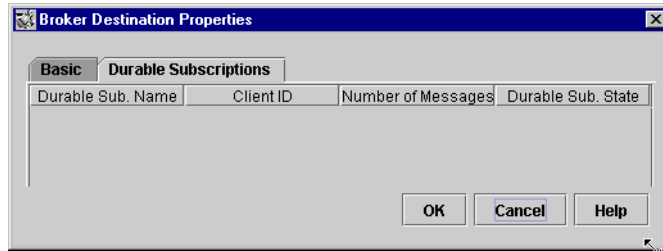


FIGURE 2-10 Durable Subscriptions Panel

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

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

The Durable Subscriptions tab is disabled for queue destinations.

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

Purging Messages From a Physical Destination

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

▼ To Purge Messages From a Physical Destination

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

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

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

- 3 **Choose Purge Messages** from the Actions menu.

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

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

Deleting a Physical Destination

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

▼ To Delete a Physical Destination

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

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

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

- 3 **Choose Delete from the Edit menu.**

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

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

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

Working With Object Stores

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

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

Adding an Object Store

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

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

▼ To Add an Object Store to the Administration Console

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

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

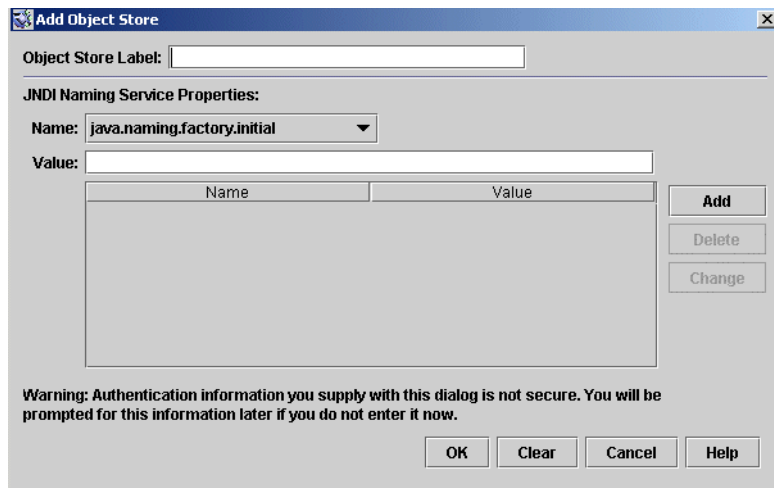


FIGURE 2–11 Add Object Store Dialog Box

- 2 Enter a name for the object store in the Object Store Label field.
This provides a label that identifies the object store in the Administration Console.
For this exercise, type in the name MyObjectStore.
- 3 Enter the JNDI attribute values to be used for looking up administered objects:
 - a. Select the name of the attribute you wish to specify from the Name pull-down menu.
 - b. Type the value of the attribute into the Value field.
 - c. Click the Add button to add the specified attribute value.
The property and its value will appear in the property summary pane.

Repeat steps “Adding an Object Store” on page 53 to “Adding an Object Store” on page 53 for as many attributes as you need to set.

For this exercise, set the `java.naming.factory.initial` attribute to `com.sun.jndi.fscontext.RefFSContextFactory`

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

`file:///C:/Temp`

(or `file:///tmp` on the Solaris or Linux platforms). These are the only attributes you need to set for a file-system object store; see “LDAP Server Object Stores” on page 153 for information on the attribute values needed for an LDAP store.

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

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

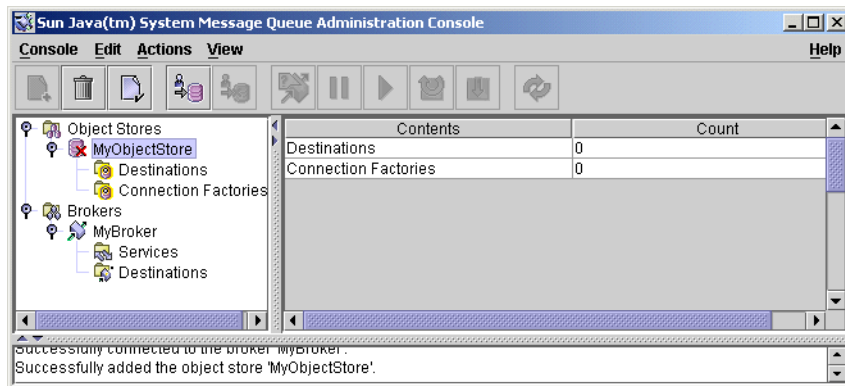


FIGURE 2-12 Object Store Displayed in Administration Console Window

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

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

Connecting to an Object Store

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

▼ To Connect to an Object Store

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

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

Working With Administered Objects

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

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

Adding a Connection Factory

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

▼ To Add a Connection Factory to an Object Store

- 1 Make sure the object store is connected to the Administration Console (see [“Connecting to an Object Store” on page 55](#)).
- 2 Click on the **Connection Factories** item under the object store's name in the Administration Console window's navigation pane and choose **Add Connection Factory Object** from the Actions menu.

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

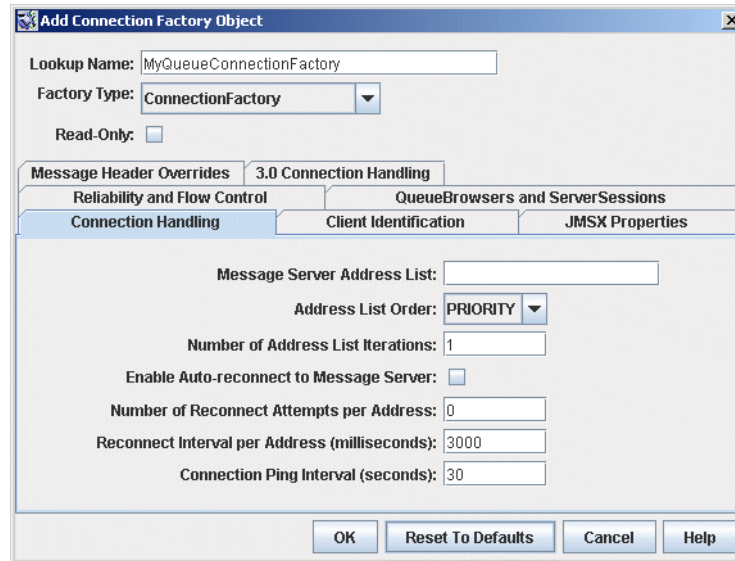


FIGURE 2-13 Add Connection Factory Object Dialog Box

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

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

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

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

For this exercise, choose `QueueConnectionFactory`.

5 Click the Connection Handling tab.

The Connection Handling panel will appear, as shown in [Figure 2-13](#).

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

The address list may consist of a single broker or (in the case of a broker cluster) multiple brokers. For each broker, it specifies information such as the broker's connection service, host name, and port number. The exact nature and syntax of the information to be specified varies, depending on the connection service to be used; see "[Connection Handling](#)" on page 293 for specifics.

For this exercise, there is no need to type anything into the Message Server Address List field, since the sample application `HelloWorldMessageJNDI` expects the connection factory to use the

standard address list attributes to which it is automatically configured by default (connection service `jms`, host name `localhost`, and port number `7676`).

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

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

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

8 If appropriate, click the Read-Only checkbox.

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

For this exercise, do not check Read-Only.

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

The new connection factory will appear in the result pane.

Adding a Destination

A *destination* administered object represents a physical destination on a broker, enabling clients to send messages to that physical destination independently of provider-specific configurations and naming syntax. When a client sends a message addressed via the administered object, the broker will deliver the message to the corresponding physical destination, if it exists. If no such physical destination exists, the broker will create one automatically if auto-creation is enabled, as described under [“Creating a Physical Destination” on page 48](#), and deliver the message to it; otherwise, it will generate an error signaling that the message cannot be delivered.

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

▼ To Add a Destination to an Object Store

- 1 Make sure the object store is connected to the Administration Console (see [“Connecting to an Object Store” on page 55](#)).**

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

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

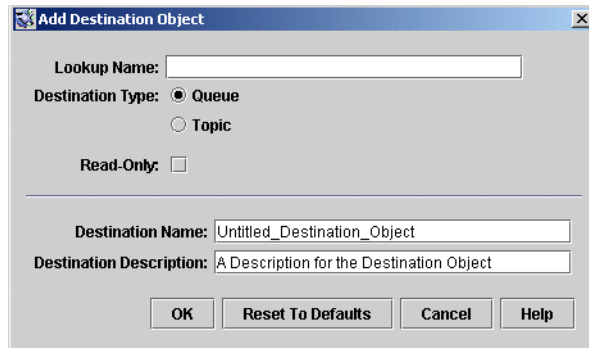


FIGURE 2-14 Add Destination Object Dialog Box

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

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

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

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

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

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

This is the name you specified when you added the physical destination to the broker (see [“Working With Physical Destinations” on page 48](#)).

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

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

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

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

Example destination for MQ Admin Guide tutorial

7 If appropriate, click the Read-Only checkbox.

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

For this exercise, do not check Read-Only.

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

The new destination object will appear in the result pane, as shown in [Figure 2–15](#).

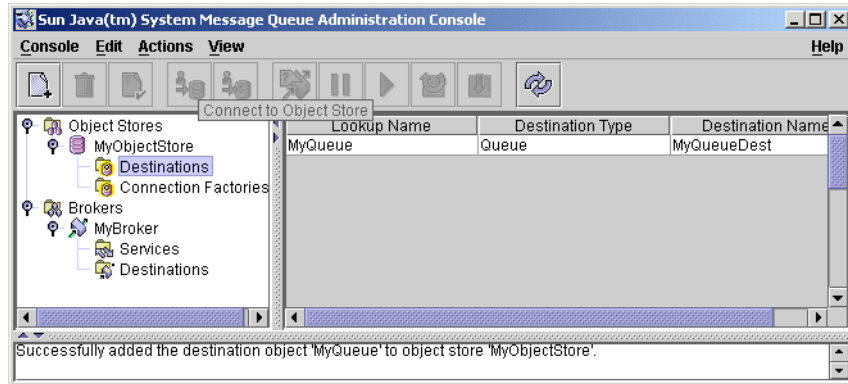


FIGURE 2–15 Destination Object Displayed in Administration Console Window

Viewing Administered Object Properties

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

▼ To View or Modify the Properties of an Administered Object

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

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

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

3 Choose Properties from the Actions menu.

The Connection Factory Object Properties or Destination Object Properties dialog box will appear, similar to the Add Connection Factory Object ([Figure 2–13](#)) or Add Destination Object ([Figure 2–14](#)) dialog. You can use this dialog box to change the selected object's configuration

attributes. Note, however, that you cannot change the object's lookup name; the only way to do this is the delete the object and then add a new administered object with the desired lookup name.

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

Deleting an Administered Object

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

▼ To Delete an Administered Object

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

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

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

- 3 **Choose Delete from the Edit menu.**

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

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

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

Running the Sample Application

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

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

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

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

▼ To Run the Sample Application

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

- On Solaris:

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

- On Linux:

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

- On Windows:

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

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

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

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

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

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

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

- On Solaris or Linux:

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

- On Windows:

```
java HelloWorldMessageJNDI
```

If the application runs successfully, you should see the output shown below.

```
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

Administrative Tasks

- [Chapter 3](#)
- [Chapter 4](#)
- [Chapter 5](#)
- [Chapter 6](#)
- [Chapter 7](#)
- [Chapter 8](#)
- [Chapter 9](#)
- [Chapter 10](#)
- [Chapter 11](#)
- [Chapter 12](#)

Starting Brokers and Clients

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

This chapter contains the following sections:

- “Preparing System Resources” on page 67
- “Starting Brokers” on page 68
- “Removing Brokers” on page 72
- “Starting Clients” on page 73

Preparing System Resources

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

Synchronizing System Clocks

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

Configure your systems to run a time synchronization protocol, such as Simple Network Time Protocol (SNTP). Time synchronization is generally supported by the `xntpd` daemon on Solaris

and Linux, and by the W32Time service on Windows. (See your operating system documentation for information about configuring this service.) After the broker is running, avoid setting the system clock backward.

Setting the File Descriptor Limit

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

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

Starting Brokers

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

Starting Brokers Interactively

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

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

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

```
imqbrokerd
```

This starts a broker instance named `imqbroker` on the local machine, with the Port Mapper at the default port of 7676 (see [“Port Mapper” on page 76](#)).

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

```
imqbrokerd -name myBroker
```

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

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

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

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

See [“Broker Utility” on page 244](#) for complete information on the syntax, subcommands, and options of the `imqbrokerd` command. For a quick summary of this information, enter the command

```
imqbrokerd -help
```

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

```
imqbrokerd -license try
```

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

Starting Brokers Automatically

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

Automatic Startup on Solaris and Linux

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

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

Automatic Startup on Windows

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

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

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

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

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

To pass startup options to the broker, use the `-args` argument to the `imqsvcadmin` command. This works the same way as the `imqbrokerd` command's `-D` option, as described under [“Starting Brokers” on page 68](#). Use the Command utility (`imqcmd`) to control broker operations as usual.

See [“Service Administrator Utility” on page 260](#) for complete information on the syntax, subcommands, and options of the `imqsvcadmin` command.

Reconfiguring the Broker Service

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

▼ To Reconfigure a Broker Running as a Windows Service

- 1 Stop the service.
 - a. From the Settings submenu of the Windows Start menu, choose Control Panel.
 - b. Open the Administrative Tools control panel.

- c. Run the Services tool by selecting its icon and choosing Open from the File menu or the pop-up context menu, or simply by double-clicking the icon.
- d. Under Services (Local), select the Message Queue Broker service and choose Properties from the Action menu.

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

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

2 Remove the service.

On the command line, enter the command

```
imqsvcadmin remove
```

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

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

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

Using an Alternative Java Runtime

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

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

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

Displaying Broker Service Startup Options

To determine the startup options for the broker service, use the `query` option to the `imqsvcadmin` command, as shown in [Example 3–1](#).

EXAMPLE 3-1 Displaying Broker Service Startup Options

```
imqsvcadmin query
```

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

Troubleshooting Service Startup Problems

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

▼ To See Logged Service Error Events

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

Removing Brokers

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

Removing a Broker on Solaris or Linux

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

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

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

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


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

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

See “[Broker Utility](#)” on page 244 for complete information on the syntax, subcommands, and options of the `imqbrokerd` command. For a quick summary of this information, enter the command

Removing a Windows Broker Service

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

```
imqcmd shutdown bkr
```

to shut down the broker, followed by

```
imqsvcadm remove
```

to remove the service.

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

Restart your computer after removing the broker service.

Starting Clients

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

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

```
java clientAppName
```

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

The application's documentation should provide information on attribute values that the application sets; you may want to override some of these from the command line. You may also want to specify attributes on the command line for any Java client that uses a Java Naming and Directory Interface (JNDI) lookup to find its connection factory. If the lookup returns a

connection factory that is older than the application, the connection factory may lack support for more recent attributes. In such cases, Message Queue sets those attributes to default values; if necessary, you can use the command line to override these default values.

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

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

The value for *attribute* must be a connection factory administered object attribute, as described in [Chapter 16](#). If there is a space in the value, put quotation marks around the *attribute=value* part of the command line.

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

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

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

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

Configuring a Broker

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

The chapter contains the following sections:

- “Broker Services” on page 75
- “Setting Broker Properties” on page 88
- “Configuring a Persistent Data Store” on page 91

For full reference information about broker configuration properties, see [Chapter 14](#)

Broker Services

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

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

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

Connection Services

Message brokers can offer various *connection services* supporting both application and administrative clients, using a variety of transport protocols. Broker configuration properties related to connection services are listed under “[Connection Properties](#)” on page 263.

[Table 4–1](#) shows the available connection services which are distinguished by two characteristics:

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

TABLE 4–1 Message Queue Connection Services

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

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

Each connection service also supports specific authentication and authorization features; see “[Security Services](#)” on page 82 for more information.

Port Mapper

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

Alternatively, you can override the Port Mapper and explicitly assign a static port number to a connection service, using the `imq.serviceName.protocolType.port` configuration property (where `serviceName` and `protocolType` identify the specific connection service, as shown in [Table 4-1](#)). (Only the `jms`, `ssljms`, `admin`, and `ssladmin` connection services can be configured this way; the `httpjms` and `httpsjms` services use different configuration properties, described in [Appendix C](#)) Static ports are generally used only in special situations, however, such as in making connections through a firewall (see [“Connecting Through a Firewall” on page 151](#)), and are not recommended for general use.

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

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

Thread Pool Management

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

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

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

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

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

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

Routing Services

Once clients are connected to the broker, the routing and delivery of messages can proceed. In this phase, the broker is responsible for creating and managing different types of physical destinations, ensuring a smooth flow of messages, and using resources efficiently. You can use the broker configuration properties described under [“Routing Properties” on page 265](#) to manage these tasks in a way that suits your application's needs.

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

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

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

As a convenience during application development and testing, you can configure a message broker to create new physical destinations automatically whenever a message producer or consumer attempts to access a nonexistent destination. The broker properties summarized in [Table 14–3](#) parallel the ones just described, but apply to such *auto-created destinations* instead of administratively created ones.

- **System memory thresholds** define levels of memory usage at which the broker takes increasingly serious action to prevent memory overload. Four such usage levels are defined:
 - **Green:** Plenty of memory is available.
 - **Yellow:** Broker memory is beginning to run low.
 - **Orange:** The broker is low on memory.
 - **Red:** The broker is out of memory.

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

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

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

Persistence Services

For a broker to recover in case of failure, it needs to re-create the state of its message delivery operations. To do this, the broker must save state information to a *persistent data store*. When the broker restarts, it uses the saved data to re-create destinations and durable subscriptions,

recover persistent messages, roll back open transactions, and rebuild its routing table for undelivered messages. It can then resume message delivery.

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

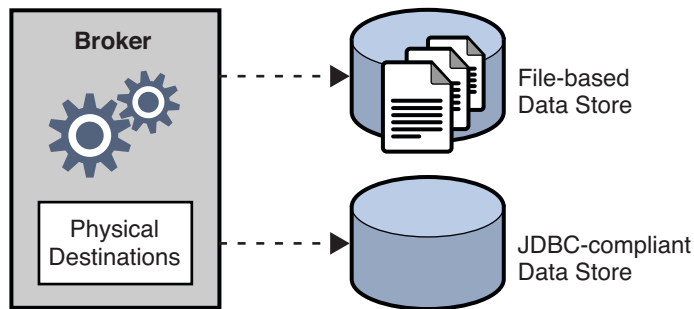


FIGURE 4–1 Persistent Data Storage

File-Based Persistence

By default, Message Queue uses a file-based persistent data store, in which individual files store persistent data such as messages, destinations, durable subscriptions, and transactions. Broker configuration properties related to file-based persistence are listed under “[File-Based Persistence](#)” on page 270.

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

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

(See [Appendix A](#) for the location of the instances directory.) Each destination on the broker has its own subdirectory holding messages delivered to that destination.

Note – Because the persistent data store can contain messages of a sensitive or proprietary nature, you should secure the `.../instances/instanceName/fs350/` directory against unauthorized access; see “[Securing Persistent Data](#)” on page 93.

All persistent data other than messages is stored in separate files: one file for destinations, one for durable subscriptions, and one for transaction state information. Most messages are stored

in a single file consisting of variable-sized records. You can compact this file to alleviate fragmentation as messages are added and removed (see [“Compacting Physical Destinations” on page 121](#)). In addition, messages above a certain threshold size are stored in their own individual files rather than in the variable-sized record file. You can configure this threshold size with the broker property `imq.persist.file.message.max_record_size`.

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

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

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

JDBC-Based Persistence

Instead of using file-based persistence, you can set up a broker to access any data store accessible through a JDBC-compliant driver. This involves setting the appropriate JDBC-related broker configuration properties and using the Database Manager utility (`imqdbmgr`) to create a database with the proper schema. See [“Configuring a JDBC-Based Store” on page 91](#) for specifics.

The properties for configuring a broker to use a JDBC database are listed under [“JDBC-Based Persistence” on page 271](#). You can specify these properties either in the instance configuration file (`config.properties`) of each broker instance or by using the `-D` command line option to the Broker utility (`imqbrokerd`) or the Database Manager utility (`imqdbmgr`).

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

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

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

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

Security Services

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

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

As a Message Queue administrator, you are responsible for setting up the information the broker needs to authenticate users and authorize their actions. The broker properties pertaining to security services are listed under “[Security Properties](#)” on [page 274](#). The boolean property `imq.accesscontrol.enabled` acts as a master switch that controls whether access control is applied on a brokerwide basis; for finer control, you can override this setting for a particular

connection service by setting the `imq.serviceName.accesscontrol.enabled` property, where `serviceName` is the name of the connection service, as shown in [Table 4-1](#): for example, `imq.httpjms.accesscontrol.enabled`.

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

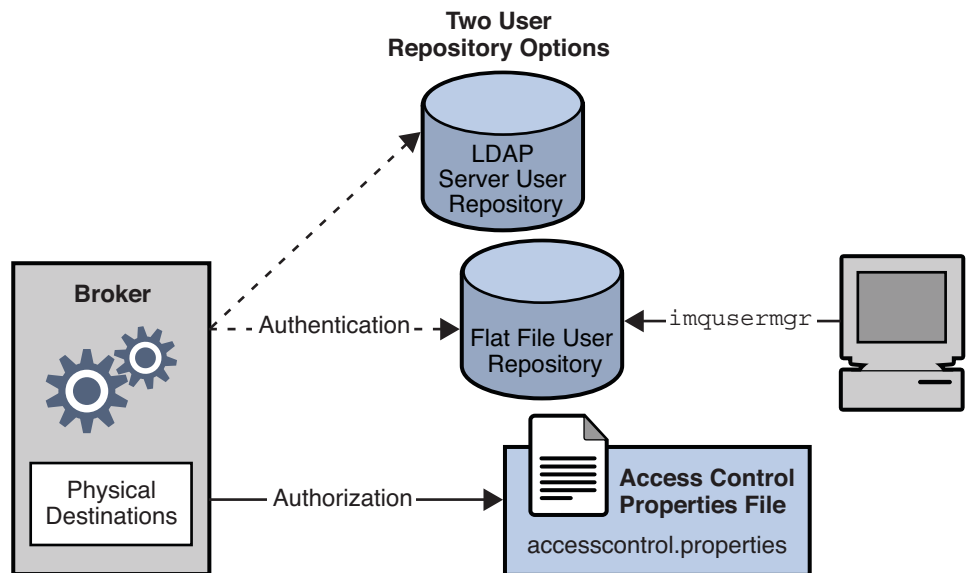


FIGURE 4-2 Security Support

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

- If you choose a flat-file repository, you must use the Message Queue User Manager utility (`imqusermgr`) to manage the repository. This option is built-in and easy to use.
- If you want to use an existing LDAP server, you use the tools provided by the LDAP vendor to populate and manage the user repository. You must also set properties in the broker's instance configuration file to enable the broker to query the LDAP server for information about users and groups.

The broker's `imq.authentication.basic.user_repository` property specifies which type of repository to use. In general, an LDAP repository is preferable if scalability is important or if you need the repository to be shared by different brokers (if you are using broker clusters, for instance). See [“User Authentication” on page 127](#) for more information on setting up a flat-file or LDAP user repository.

Authentication

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

As described under [“Password Files” on page 149](#), you can choose to put your passwords in a *password file* instead of being prompted for them interactively. The boolean broker property `imq.passfile.enabled` controls this option. If this property is true, the `imq.passfile.dirpath` and `imq.passfile.name` properties give the directory path and file name for the password file. The `imq.imqcmd.password` property (which can be embedded in the password file) specifies the password for authenticating an administrative user to use the Command utility (`imqcmd`) for managing brokers, connection services, connections, physical destinations, durable subscriptions, and transactions.

If you are using an LDAP-based user repository, there are a whole range of broker properties available for configuring various aspects of the LDAP lookup. The address (host name and port number) of the LDAP server itself is specified by `imq.user_repository.ldap.server`. The `imq.user_repository.ldap.principal` property gives the distinguished name for binding to the LDAP repository, while `imq.user_repository.ldap.password` supplies the associated password. Other properties specify the directory bases and optional JNDI filters for individual user and group searches, the provider-specific attribute identifiers for user and group names, and so forth; see [“Security Properties” on page 274](#) for details.

Authorization

Once authenticated, a user can be authorized to perform various Message Queue-related activities. As a Message Queue administrator, you can define user groups and assign individual users membership in them. The default access control properties file explicitly refers to only one group, `admin` (see [“Groups” on page 130](#)). A user in this group has connection permission for the `admin` connection service, which allows the user to perform administrative functions such as creating destinations and monitoring and controlling a broker. A user in any other group that you define cannot, by default, get an `admin` service connection.

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

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

Encryption

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

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

The `imq.audit.enabled` broker property controls the logging of audit records to the Message Queue broker log file; see [“Audit Logging” on page 152](#) for more information.

Monitoring Services

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

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

The general scheme is illustrated in [Figure 4–3](#). Broker properties for configuring the monitoring services are listed under [“Monitoring Properties” on page 278](#).

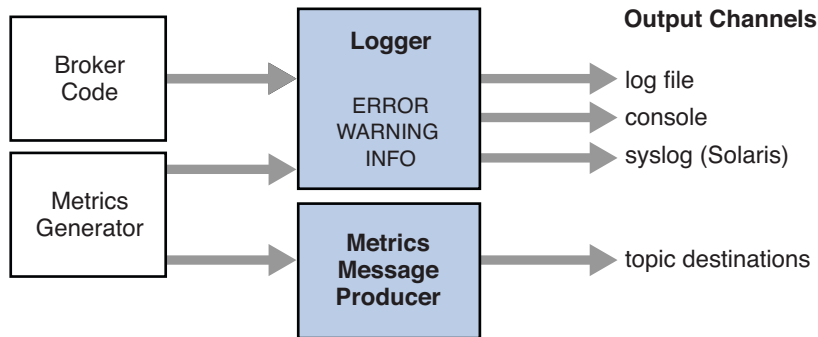


FIGURE 4-3 Monitoring Support

Metrics Generator

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

Logger

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

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

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

See “[Monitoring Properties](#)” on page 278 for additional broker properties related to logging, and “[Configuring and Using Broker Logging](#)” on page 181 for further details about how to configure the Logger and how to use it to obtain performance information.

Metrics Message Producer (Enterprise Edition)

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

TABLE 4–2 Metric Topic Destinations

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

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

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

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

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

Setting Broker Properties

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

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

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

Configuration Files

Broker configuration files contain property settings for configuring a broker. They are kept in a directory whose location depends on the operating system platform you are using; see [Appendix A](#) for details. The directory stores the following files:

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

In addition, each individual broker instance has its own *instance configuration file*, as described below. If you connect broker instances in a cluster, you may also need to use a *cluster configuration file* to specify configuration information for the cluster; see “[Cluster Configuration Properties](#)” on page 282 for more information.

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

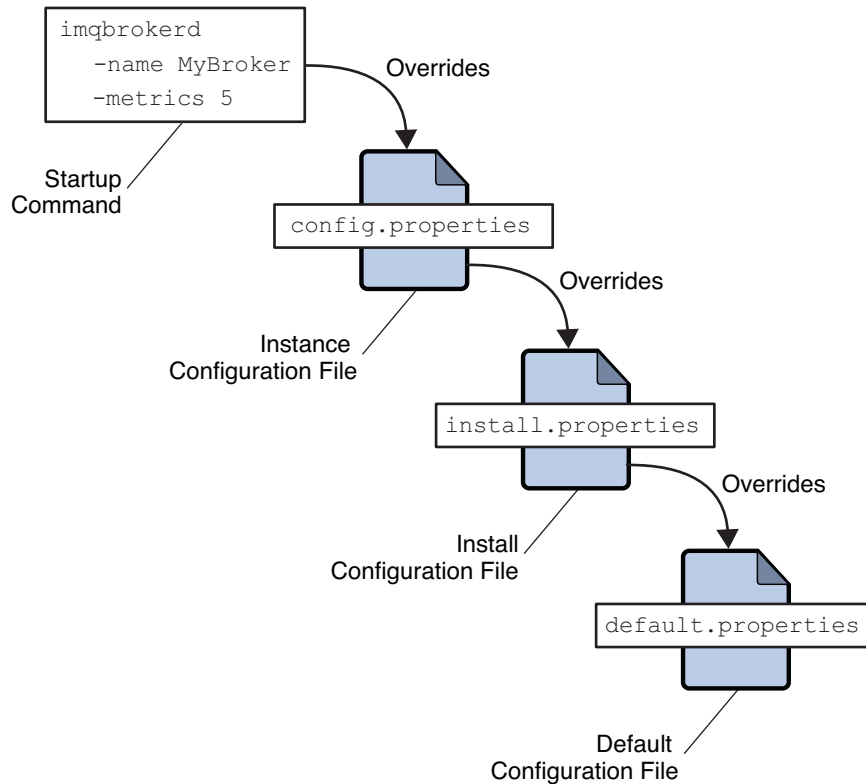


FIGURE 4-4 Broker Configuration Files

Editing the Instance Configuration File

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

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

(See [Appendix A](#) for the location of the instances directory.) If the file does not yet exist, you must use the `-name` option when starting the broker (see “[Broker Utility](#)” on page 244), to specify an instance name that Message Queue can use to create the file.

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

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

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

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

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

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

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

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

See [“Broker Services” on page 75](#) and [Chapter 14](#) for information on the available broker configuration properties and their default values.

Setting Configuration Options from the Command Line

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

At startup time, you use the Broker utility (`imqbrokerd`) to start a broker instance. Using the command's `-D` option, you can specify any broker configuration property and its value; see [“Starting Brokers” on page 68](#) and [“Broker Utility” on page 244](#) for more information. If you start the broker as a Windows service, using the Service Administrator utility (`imqsvcadmin`), you use the `-args` option to specify startup configuration properties; see [“Service Administrator Utility” on page 260](#).

You can also change certain broker properties while a broker instance is running. To modify the configuration of a running broker, you use the Command utility's `imqcmd update bkr` command; see [“Updating Broker Properties” on page 99](#) and [“Broker Management” on page 250](#).

Configuring a Persistent Data Store

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

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

- [“Configuring a File-Based Store” on page 91](#)
- [“Configuring a JDBC-Based Store” on page 91](#)
- [“Securing Persistent Data” on page 93](#)

Configuring a File-Based Store

A file-based data store is automatically created when you create a broker instance. The store is located in the broker's instance directory; see [Appendix A](#) for the exact location.

By default, Message Queue performs asynchronous write operations to disk. The operating system can buffer these operations for efficient performance. However, if an unexpected system failure should occur between write operations, messages could be lost. To improve reliability (at the cost of reduced performance), you can set the broker property `imq.persist.file.sync` to write data synchronously instead. For further discussion about this property, see [“File-Based Persistence” on page 80](#) and [Table 14-5](#).

When you start a broker instance, you can use the `imqbrokerd` command's `-reset` option to clear the file system store. For more information about this option and its suboptions, see [“Broker Utility” on page 244](#).

Configuring a JDBC-Based Store

To configure a broker to use JDBC-based persistence, you set JDBC-related properties in the broker's instance configuration file and create the appropriate database schema. The Message Queue Database Manager utility (`imqdbmgr`) uses your JDBC driver and the broker configuration properties to create and manage the database. You can also use the Database Manager to delete corrupted tables from the database or if you want to use a different database as a data store. See [“Database Manager Utility” on page 257](#) for more information.

Note – Example configurations for Oracle and PointBase database products are available. The location of these files is platform-dependent, and is listed under “Example applications and configurations” in the relevant tables of [Appendix A](#). In addition, examples for PointBase embedded version, PointBase server version, and Oracle are provided as commented-out values in the instance configuration file, `config.properties`.

▼ To Configure a JDBC-Based Data Store

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

The relevant properties are discussed under “[JDBC-Based Persistence](#)” on page 81 and listed in [Table 14–6](#). In particular, you must set the broker’s `imq.persist.store` property to `jdbc` (see “[Persistence Properties](#)” on page 270).

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

- On Solaris:

```
/usr/share/lib/imq/ext/
```

- On Linux:

```
/opt/sun/mq/share/lib/
```

- On Windows:

```
IMQ_VARHOME\\lib\\ext
```

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

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

The following command creates a symbolic link instead:

```
% ln -s j2eeSDKID/lib/pointbase/pointbase.jar /usr/share/lib/imq/ext
```

3 Create the database schema needed for Message Queue persistence.

Use the `imqdbmgr create all` command (for an embedded database) or the `imqdbmgr create tbl` command (for an external database); see “[Database Manager Utility](#)” on page 257.

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

- On Solaris:

```
cd /usr/bin
```

- On Linux:

```
cd /opt/sun/mq/bin
```

- On Windows:

```
cd IMQ_HOME\bin
```

b. Enter the `imqdbmgr` command:

```
imqdbmgr create all
```

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

```
.../instances/ instanceName/dbstore/databaseName
```

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

Securing Persistent Data

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

Securing a File-Based Store

A broker using file-based persistence writes persistent data to a flat-file data store whose location is platform-dependent (see [Appendix A](#)):

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

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

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

- On Solaris and Linux, the directory's permissions are determined by the file mode creation mask (`umask`) of the user who started the broker instance. Hence, permission to start a broker instance and to read its persistent files can be restricted by setting the mask appropriately. Alternatively, an administrator (superuser) can secure persistent data by setting the permissions on the `instances` directory to `700`.

- On Windows, the directory's permissions can be set using the mechanisms provided by the Windows operating system. This generally involves opening a Properties dialog for the directory.

Securing a JDBC-Based Store

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

The user name and password required to open a database connection by a broker can be provided as broker configuration properties. However it is more secure to provide them as command line options when starting up the broker, using the `imqbrokerd` command's `-dbuser` and `-dbpassword` options (see [“Broker Utility” on page 244](#)).

For an embedded database that is accessed directly by the broker via the database's JDBC driver, security is usually provided by setting file permissions on the directory where the persistent data will be stored, as described above under [“Securing a File-Based Store” on page 93](#). To ensure that the database is readable and writable by both the broker and the Database Manager utility, however, both should be run by the same user.

Managing a Broker

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

- “Prerequisites” on page 96
- “Using the `imqcmd` Utility” on page 96
- “Displaying Broker Information” on page 98
- “Updating Broker Properties” on page 99
- “Pausing and Resuming a Broker” on page 100
- “Shutting Down and Restarting a Broker” on page 101
- “Displaying Broker Metrics” on page 102
- “Managing Connection Services” on page 103
- “Getting Information About Connections” on page 107
- “Managing Durable Subscriptions” on page 108
- “Managing Transactions” on page 110

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

- Management of physical destinations on the broker. For information about topics such as how to create, display, update and destroy physical destinations, and how to use the dead message queue, see [Chapter 6](#)
- 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](#)

Prerequisites

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

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

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

If you are setting up a production system, you must set up authentication and authorization for administrative users. See [Chapter 7](#) 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 [“Message Encryption” on page 141](#).

Using the `imqcmd` Utility

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

Reference information about the syntax, subcommands, and options of the `imqcmd` command is in [Chapter 13](#). Reference information for managing physical destinations is in a separate chapter, [Chapter 15](#).

Displaying Help

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

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

```
imqcmd -H
```

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

Displaying the Product Version

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

```
imqcmd -v
```

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

Specifying the User Name and Password

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

Specifying the User Name

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

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

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

Specifying the Password

Specify the password using one of the following methods:

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

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

Specifying the Broker Name and Port

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

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

Examples

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

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

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

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

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

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

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

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

Displaying Broker Information

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

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

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

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

For example:

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

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

Version	3.6
Instance Name	imqbroker
Primary Port	7676
Current Number of Messages in System	0
Current Total Message Bytes in System	0
Current Number of Messages in Dead Message Queue	0
Current Total Message Bytes in Dead Message Queue	0
Log Dead Messages	true
Truncate Message Body in Dead Message Queue	false
Max Number of Messages in System	unlimited (-1)
Max Total Message Bytes in System	unlimited (-1)
Max Message Size	70m
Auto Create Queues	true
Auto Create Topics	true
Auto Created Queue Max Number of Active Consumers	1
Auto Created Queue Max Number of Backup Consumers	0
Cluster Broker List (active)	
Cluster Broker List (configured)	
Cluster Master Broker	
Cluster URL	
Log Level	INFO
Log Rollover Interval (seconds)	604800
Log Rollover Size (bytes)	unlimited (-1)

Updating Broker Properties

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

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

- `imq.system.max_size`
- `imq.message.max_size`
- `imq.portmapper.port`

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

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

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

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

The properties are described in [Chapter 14](#)

Pausing and Resuming a Broker

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

Pausing a Broker

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

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

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

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

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

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

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

You can also pause individual connection services and individual physical destinations. For more information, see [“Pausing and Resuming a Connection Service” on page 106](#) and [“Pausing and Resuming Physical Destinations” on page 119](#).

Resuming a Broker

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

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

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

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

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

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

Shutting Down and Restarting a Broker

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

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

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

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

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

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

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

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

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

Displaying Broker Metrics

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

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

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

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

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

- **t~~tl~~** Displays metrics about the messages and packets flowing into and out of the broker (default metric type).
- **r~~ts~~** Displays metrics about the rate of flow of messages and packets into and out of the broker (per second).
- **c~~xn~~** 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 about the data gathered and reported by the broker, see [“Brokerwide Metrics” on page 310](#).

Managing Connection Services

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

- “Listing Connection Services” on page 103
- “Displaying Connection Service Information” on page 104
- “Updating Connection Service Properties” on page 105
- “Displaying Connection Service Metrics” on page 105
- “Pausing and Resuming a Connection Service” on page 106

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

TABLE 5–1 Message Queue Connection Services

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

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

Listing Connection Services

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

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

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

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

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

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

The command will output information like the following:

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

Displaying Connection Service Information

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

This is the syntax for the query svc subcommand:

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

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

For example:

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

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

Service Name	jms
Service State	RUNNING
Port Number	60920 (dynamic)
Current Number of Allocated Threads	0
Current Number of Connections	0
Min Number of Threads	10
Max Number of Threads	1000

Updating Connection Service Properties

You can use the update subcommand to change the value of one or more of the service properties listed in [Table 5–2](#).

TABLE 5–2 Connection Service Properties Updated by imqcmd

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

This is the syntax of the update subcommand:

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

This subcommand updates the specified attribute of the specified service running on the default broker or on a broker at the specified host and port. For a description of service attributes, see “Connection Properties” on page 263.

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

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

Displaying Connection Service Metrics

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

This is the syntax of the metrics subcommand:

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

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

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

- **t_{ttl}** Displays metrics on messages and packets flowing into and out of the broker by way of the specified connection service. (default metric type).
- **r_{ts}** 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.
- **c_{xn}** 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 312](#).

Pausing and Resuming a Connection Service

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

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

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

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

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

Pausing a service has the following effects:

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

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

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

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

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

Getting Information About Connections

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

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

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

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

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

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

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

Listing all the connections on the broker specified by:

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

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

Successfully listed connections.

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

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

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

For example:

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

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

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

Managing Durable Subscriptions

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

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

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

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

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

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

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

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

Name	Client ID	Number of Messages	Durable Sub State

myDurable	myClientID	1	INACTIVE

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

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

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

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

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

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

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

Managing Transactions

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

Each transaction has a Message Queue *transaction ID*: a 64 bit number that uniquely identifies a transaction on the broker. Distributed transactions also have a *distributed transaction ID* (XID), up to 128 bytes long, assigned by the distributed transaction manager. 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/ # Acks	Creation time
64248349708800	PREPARED	guest	4/0	1/30/02 10:08:31 AM
64248371287808	PREPARED	guest	0/4	1/30/02 10:09:55 AM

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

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

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

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

```
imqcmd query txn -n transactionID
```

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

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

```
Client ID
Connection          guest@192.18.116.219:62209->jms:62195
Creation time        1/30/02 10:08:31 AM
Number of acknowledgments 0
Number of messages   4
State                PREPARED
Transaction ID        64248349708800
User name            guest
XID
6469706F6C7369646577696E6465723130313234313431313030373230
```

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

This is the syntax of the commit subcommand:

```
imqcmd commit txn -n transactionID
```

For example:

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

This is the syntax of the rollback subcommand:

```
imqcmd rollback txn -n transactionID
```

See the `imq.transaction.autorollback` property in [Table 14–2](#) for more information.

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

Managing Physical Destinations

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

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

This chapter covers the following topics:

- “Using the Command Utility” on page 114
- “Creating a Physical Destination” on page 115
- “Listing Physical Destinations” on page 116
- “Displaying Information about Physical Destinations” on page 117
- “Updating Physical Destination Properties” on page 118
- “Pausing and Resuming Physical Destinations” on page 119
- “Purging Physical Destinations” on page 120
- “Destroying Physical Destinations” on page 120
- “Compacting Physical Destinations” on page 121
- “Using the Dead Message Queue” on page 123

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

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

Note – A client application uses a `Destination` object whenever it interacts with a physical destination. For provider-independence and portability, clients typically use administrator-created destination objects, which are called destination administered objects. You can configure administered objects for use by client applications, as described in [Chapter 8](#)

Using the Command Utility

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

Full reference information about `imqcmd`, its subcommands, and its options, is available in [Chapter 13](#).

Subcommands

[Table 6–1](#) lists the `imqcmd` subcommands whose use is described in this chapter. For reference information about these subcommands, see “[Physical Destination Management](#)” on page 252.

TABLE 6–1 Physical Destination Subcommands for the Command Utility

Subcommand and Argument	Description
<code>compact dst</code>	Compacts the file-based data store for one or more physical destinations.
<code>create dst</code>	Creates a physical destination.
<code>destroy dst</code>	Destroys a physical destination.
<code>list dst</code>	Lists physical destinations on a broker.
<code>metrics dst</code>	Displays physical destination metrics.
<code>pause dst</code>	Pauses one or more physical destinations on a broker.
<code>purge dst</code>	Purges all messages on a physical destination without destroying the physical destination.
<code>query dst</code>	Queries and displays information on a physical destination.
<code>resume dst</code>	Resumes one or more paused physical destinations on a broker.
<code>update dst</code>	Updates properties of a destination.

Creating a Physical Destination

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

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

For example, 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"
```

When creating a physical destination, you specify the following:

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

Nondefault values for the physical destination's properties.

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

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

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

- `maxNumMsgs`. Specifies the maximum number of unconsumed messages allowed in the physical destination.
- `maxTotalMsgBytes`. Specifies the maximum total amount of memory (in bytes) allowed for unconsumed messages in the physical destination.
- `limitBehavior`. Specifies how the broker responds when a memory-limit threshold is reached.
- `maxBytesPerMsg`. Specifies the maximum size (in bytes) of any single message allowed in the physical destination.

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

The following properties are used for queue destinations only:

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

See [Chapter 15](#) for full reference information about physical destination properties.

For auto-created destinations, you set default property values in the broker's instance configuration file. Reference information on auto-create properties is located in [Table 14–3](#).

Listing Physical Destinations

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

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

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

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

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

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

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

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

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

Displaying Information about Physical Destinations

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

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

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

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

The command produces output like the following:

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

On the broker specified by:

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

```
Destination Name      XQueue  
Destination Type      Queue  
Destination State     RUNNING  
Created Administratively true  
  
Current Number of Messages 0  
Current Total Message Bytes 0
```

Current Number of Producers	0
Current Number of Active Consumers	0
Current Number of Backup Consumers	0
Max Number of Messages	unlimited (-1)
Max Total Message Bytes	unlimited (-1)
Max Bytes per Message	unlimited (-1)
Max Number of Producers	100
Max Number of Active Consumers	1
Max Number of Backup Consumers	0
Limit Behavior	REJECT_NEWEST
Consumer Flow Limit	1000
Is Local Destination	false
Local Delivery is Preferred	false
Use Dead Message Queue	true

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

You can use the `update dst` subcommand to change the value of one or more properties (see [“Updating Physical Destination Properties” on page 118](#)).

Updating Physical Destination Properties

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

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

The command updates the value of the specified properties at the specified destination. The property name can be any property listed in [Table 15–1](#).

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

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

See [Chapter 15](#) 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 [“Using the Dead Message Queue” on page 123](#).

Pausing and Resuming Physical Destinations

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

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

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

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

Example:

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

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

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

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

Example:

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

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

Purging Physical Destinations

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

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

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

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

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

Examples:

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

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

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

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

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

Destroying Physical Destinations

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

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

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

Example:

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


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

You cannot destroy the dead message queue.

Compacting Physical Destinations

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

The file-based message store is structured so that messages are stored in directories corresponding to the physical destinations in which they are being held. In each physical destination's directory, most messages are stored in one file consisting of variable-sized records. (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 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 disk -u admin
```

This command produces output like the following:

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

The columns in the subcommand output have the following meaning:

TABLE 6-2 Physical Destination Disk Utilization Metrics

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

Reclaiming Unused Physical Destination Disk Space

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

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

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

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

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

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

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

▼ 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.

Using the Dead Message Queue

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

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

Configuring Use of the Dead Message Queue

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

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

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

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

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

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

Managing the Dead Message Queue

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

Dead Message Queue Properties

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

TABLE 6–3 Dead Message Queue Treatment of Standard Physical Destination Properties

Property	Unique Treatment by Dead Message Queue
<code>limitBehavior</code>	The default value for the dead message queue is <code>REMOVE_OLDEST</code> . (The default value for other queues is <code>REJECT_NEWEST</code> .) Flow control is not supported on the dead message queue.
<code>localDeliveryPreferred</code>	Does not apply to the dead message queue.
<code>maxNumMsgs</code>	The default value for the dead message queue is <code>1000</code> . The default value for other queues is <code>-1</code> (unlimited).
<code>maxNumProducers</code>	Does not apply to the dead message queue.
<code>maxTotalMsgBytes</code>	The default value for the dead message queue is 10 MB. The default value for other queues is <code>-1</code> (unlimited).
<code>isLocalOnly</code>	In a broker cluster, a dead message queue is always a global physical destination and this property is permanently set to <code>false</code> .

Message Contents

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

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

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

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

Enabling Dead Message Logging

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

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

The following command enables dead message logging:

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

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

Managing Security

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

The chapter includes the following sections:

- [“User Authentication” on page 127](#)
- [“User Authorization: The Access Control Properties File” on page 135](#)
- [“Message Encryption” on page 141](#)
- [“Password Files” on page 149](#)
- [“Audit Logging” on page 152](#)

User Authentication

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

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

The repository can be one of the following types:

- A flat-file repository that is shipped with Message Queue™
This type of user repository is very easy to use. You can populate and manage the repository using the User Manager utility (`imqusermgr`). To enable authentication, you populate the user repository with each user’s name and password and the name of the user’s group.
For more information on setting up and managing the user repository, see [“Using a Flat-File User Repository” on page 128](#)
- 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 133](#).

Using a Flat-File User Repository

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

Creating a User Repository

The flat-file user repository is instance-specific. A default user repository (named `passwd`) is automatically created for each broker instance that you start. This user repository is placed in a directory identified by the name of the broker instance with which the repository is associated (see [Appendix A](#)):

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

The repository is created with two entries. Each row of [Table 7-1](#) shows an entry.

TABLE 7-1 Initial Entries in User Repository

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

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

- The initial `guest` user entry allows clients to connect to a broker instance using the default `guest` user name and password.
- The initial `admin` user entry lets you use `imqcmd` commands to administer a broker instance using the default `admin` user name and password. You should update this initial entry to change the password (see [“Changing the Default Administrator Password” on page 132](#)).

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

User Manager Utility

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

For full reference information about the `imqusermgr` command, see [Chapter 13](#)

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

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

Note – Examples in the following sections assume the default broker instance.

Subcommands

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

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

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

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

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

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

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

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

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

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

Command Options

[Table 7–2](#) lists the options to the `imqusermgr` command.

TABLE 7-2 imqusermgr Options

Option	Description
-a <i>activeState</i>	Specifies (<code>true/false</code>) whether the user's state should be active. A value of <code>true</code> 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 <i>instanceName</i>	Specifies the broker instance name to which the command applies. If not specified, the default instance name, <code>imqbroker</code> , is assumed.
-p <i>passwd</i>	Specifies the user's password.
-g <i>group</i>	Specifies the user group. Valid values are <code>admin</code> , <code>user</code> , <code>anonymous</code> .
-s	Sets silent mode.
-u <i>userName</i>	Specifies the user name.
-v	Displays version information. Nothing else on the command line is executed.

Groups

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

- **admin group.** For broker administrators. Users who are assigned this group can, by default, configure, administer, and manage the broker. You can assign more than one user to the `admin` group.
- **user group.** For normal (non-administration) Message Queue client users. Most client users are in the `user` group. By default, users in this group can produce messages to all topics and queues, consume messages from all topics and queues, and browse messages in any queue.
- **anonymous group.** For Message Queue clients that do not want a user name that is known to the broker, possibly because the client application does not know of a real user name to use. This account is analogous to the anonymous account present in most FTP servers. You can assign only one user at a time to the `anonymous` group. You should restrict the access privileges of this group as compared to the `user` group or you should remove users from the group at deployment time.

To change a user's group, you must delete the user entry and then add another entry for the user, specifying the new group.

You cannot rename or delete these system-created groups, or create new groups. However, you can specify access rules that define the operations that the members of that group can perform. For more information, see [“User Authorization: The Access Control Properties File” on page 135](#).

User States

When you add a user to a repository, the user's state is active by default. To make the user inactive, you must use the update command. For example, the following command makes the user JoeD inactive:

```
imqusermgr update -u JoeD -a false
```

Entries for users that have been rendered inactive are retained in the repository; however, inactive users cannot open new connections. If a user is inactive and you add another user who has the same name, the operation will fail. You must delete the inactive user entry or change the new user's name or use a different name for the new user. This prevents you from adding duplicate user names.

Format of User Names and Passwords

User names and passwords must follow these guidelines:

- A user name cannot contain an asterisk (*), comma (,), colon (:), or a new-line or carriage-return character.
- A user name or password must be at least one character long.
- If a user name or password contains a space, the entire name or password must be enclosed in quotation marks.
- There is no limit on the length of passwords or user names, except for command shell restrictions on the maximum number of characters that can be entered on a command line.

Populating and Managing a User Repository

Use the add subcommand to add a user to a repository. For example, the following command adds the user Katharine with the password sesame to the default broker instance user repository.

```
imqusermgr add -u Katharine -p sesame -g user
```

Use the delete subcommand to delete a user from a repository. For example, the following command deletes the user, Bob:

```
imqusermgr delete -u Bob
```

Use the `update` subcommand to change a user's password or state. For example, the following command changes Katharine's password to `aladdin`:

```
imqusermgr update -u Katharine -p aladdin
```

To list information about one user or all users, use the `list` command. The following command shows information about the user named `isa`:

```
imqusermgr list -u isa
```

```
% imqusermgr list -u isa
```

```
User repository for broker instance: imqbroker
```

```
-----  
User Name    Group      Active State  
-----  
isa          admin      true
```

The following command lists information about all users:

```
imqusermgr list
```

```
% imqusermgr list
```

```
User repository for broker instance: imqbroker
```

```
-----  
User Name    Group      Active State  
-----  
admin        admin      true  
guest        anonymous  true  
isa          admin      true  
testuser1    user       true  
testuser2    user       true  
testuser3    user       true  
testuser4    user       false  
testuser5    user       false
```

Changing the Default Administrator Password

For the sake of security, you should change the default password of `admin` to one that is known only to you. The following command changes the default administrator password for the `mybroker` broker instance from `admin` to `grandpoobah`.

```
imqusermgr update mybroker -u admin -p grandpoobah
```

You can quickly confirm that this change is in effect by running any of the command line tools when the broker instance is running. For example, the following command will prompt you for a password:

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

Entering the new password (grandpoobah) should work; the old password should fail.

After changing the password, you should supply the new password any time you use any of the Message Queue administration tools, including the Administration Console.

Using an LDAP Server for a User Repository

To use an LDAP server for a user repository, you perform the following tasks:

- Editing the instance configuration file
- Setting up access control for administrators

Editing the Instance Configuration File

To have a broker use a directory server, you set the values for certain properties in the broker instance configuration file, `config.properties`. These properties enable the broker instance to query the LDAP server for information about users and groups whenever a user attempts to connect to the broker instance or perform messaging operations.

The instance configuration file is located in a directory under the broker instance directory. The path has the following format:

```
.../instances/instanceName
```

```
/props/config.properties
```

For information about the operating system-specific location of instance directories, see [Appendix A](#)

▼ To Edit the Configuration File to Use an LDAP Server

- 1 **Specify that you are using an LDAP user repository by setting the following property:**

```
imq.authentication.basic.user_repository=ldap
```

- 2 **Set the `imq.authentication.type` property to determine whether a password should be passed from client to broker in base-64 (`basic`) or MD5 (`digest`) encoding. When using an LDAP directory server for a user repository, you must set the authentication type to `basic`. For example,**

```
imq.authentication.type=basic
```

- 3 **You must also set the broker properties that control LDAP access. These properties are stored in a broker's instance configuration file. The properties are discussed under [“Security Services” on page 82](#) and summarized under [“Security Properties” on page 274](#).**

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 [“User Authorization: The Access Control Properties File” on page 135](#).**
- 5 **If you want the broker to communicate with the LDAP directory server over SSL during connection authentication and group searches, you need to activate SSL in the LDAP server and then set the following properties in the broker configuration file:**

- Specify the port used by the LDAP server for SSL communications. For example:

```
imq.user_repository.ldap.server=myhost:7878
```

- Set the broker property `imq.user_repository.ldap.ssl.enabled` to `true`.

When employing multiple LDAP directory servers, use `ldap://` to specify each additional directory server. For example:

```
imq.user_repository.ldap.server= myHost:7878 ldap:// otherHost:7878 ...
```

Separate each additional directory server with a space. All directory servers in the list must use the same values for other LDAP-related properties.

Setting Up Access Control for Administrators

To create administrative users, you use the access control properties file to specify users and groups that can create ADMIN connections. These users and groups must be predefined in the LDAP directory.

Any user or group who can create an ADMIN connection can issue administrative commands.

▼ To Set Up an Administrative User

- 1 **Enable the use of the access control file by setting the broker property**

`imq.accesscontrol.enabled` to `true`, which is the default value.

The `imq.accesscontrol.enabled` property enables use of the access control file.

- 2 **Open the access control file, `accesscontrol.properties`. The location for the file is listed in [Appendix A](#)**

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] ...]
```

User Authorization: The Access Control Properties File

An *access control properties file* (ACL file) contains rules that specify the operations that users and groups of users can perform. You edit the ACL file to restrict operations to certain users and groups. You can use a different ACL file for each broker instance.

The ACL file is used whether user information is placed in a flat-file user repository or in an LDAP user repository. A broker checks its ACL file when a client application performs one of the following operations:

- Creates a connection
- Creates a producer
- Creates a consumer
- Browses a queue

The broker checks the ACL file to determine whether the user that generated the request, or a group to which the user belongs, is authorized to perform the operation.

If you edit an ACL file, the new settings take effect the next time the broker checks the file to verify authorization. You need not restart the broker after editing the file.

Creating an Access Control Properties File

The ACL file is instance specific. Each time you start a broker instance, a default file named `accesscontrol.properties` is created in the instance directory. The path to the file has the following format (see [Appendix A](#)):

```
.../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
<i>resourceType</i>	One of the following: connection, queue or topic.
<i>resourceVariant</i>	An instance of the type specified by <i>resourceType</i> . For example, myQueue. The wild card character (*) may be used to mean all connection service types or all physical destinations.
<i>operation</i>	Value depends on the kind of access rule being formulated.
<i>access</i>	One of the following: allow or deny.
<i>principalType</i>	One of the following: user or group. For more information, see “Groups” on page 130.
<i>principals</i>	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 <i>principalType</i> is user; it may be a single group or a list of groups (comma delimited list) if the <i>principalType</i> is group. The wild card character (*) may be used to represent all users or all groups.

Here are some examples of access rules:

- The following rule means that all users may send a message to the queue named q1.

```
queue.q1.produce.allow.user=*
```

- The following rule means that any user may send messages to any queue.

```
queue.*.produce.allow.user=*
```

Note – To specify non-ASCII user, group, or destination names, use Unicode escape (\uXXXX) notation. If you have edited and saved the ACL file with these names in a non-ASCII encoding, you can convert the file to ASCII with the Java native2ascii tool. For more detailed information, see

<http://java.sun.com/j2se/1.4/docs/guide/intl/faq.html>

How Permissions are Computed

When there are multiple access rules in the file, permissions are computed as follows:

- Specific access rules override general access rules. After applying the following two rules, all users can send to all queues, but Bob cannot send to tq1.

```
queue.*.produce.allow.user=*
queue.tq1.produce.deny.user=Bob
```

- Access given to an explicit *principal* overrides access given to a ** principal*. The following rules deny Bob the right to produce messages to tq1, but allow everyone else to do it.

```
queue.tq1.produce.allow.user=*  
queue.tq1.produce.deny.user=Bob
```

- The ** principal* rule for users overrides the corresponding ** principal* for groups. For example, the following two rules allow all authenticated users to send messages to tq1.

```
queue.tq1.produce.allow.user=*  
queue.tq1.produce.deny.group=*
```

- Access granted a user overrides access granted to the user's group. In the following example, even if Bob is a member of User, he cannot produce messages to tq1. All other members of User will be able to do so.

```
queue.tq1.produce.allow.group=User  
queue.tq1.produce.deny.user=Bob
```

- Any access permission not explicitly granted through an access rule is implicitly denied. For example, if the ACL file contains no access rules, all users are denied all operations.
- Deny and allow permissions for the same user or group cancel themselves out. For example, the following two rules cause Bob to be unable to browse q1:

```
queue.q1.browse.allow.user=Bob  
queue.q1.browse.deny.user=Bob
```

The following two rules prevent the group User from consuming messages at q5.

```
queue.q5.consume.allow.group=User  
queue.q5.consume.deny.group=User
```

- When multiple same left-hand rules exist, only the last entry takes effect.

Access Control for Connection Services

The connection access control section in the ACL properties file contains access control rules for the broker's connection services. The syntax of connection access control rules is as follows:

```
connection.resourceVariant.  
access.principalType=  
principals
```

Two values are defined for *resourceVariant*: NORMAL and ADMIN. These predefined values are the only types of connection services to which you can grant access.

The default ACL properties file gives all users access to NORMAL connection services and gives users in the group admin access to ADMIN connection services:

```
connection.NORMAL.allow.user=*
connection.ADMIN.allow.group=admin
```

If you are using a file-based user repository, the default group `admin` is created by the User Manager utility. If you are using an LDAP user repository, you can do one of the following to use the default ACL properties file:

- Define a group called `admin` in the LDAP directory.
- Replace the name `admin` in the ACL properties file with the names of one or more groups that are defined in the LDAP directory.

You can restrict connection access privileges. For example, the following rules deny Bob access to `NORMAL` but allow everyone else:

```
connection.NORMAL.deny.user=Bob
connection.NORMAL.allow.user=*
```

You can use the asterisk (*) character to specify all authenticated users or groups.

The way that you use the ACL properties file to grant access to `ADMIN` connections differs for file-based user repositories and LDAP user repositories, as follows:

- **File-based user repository**
 - If access control is disabled, users in the group `admin` have `ADMIN` connection privileges.
 - If access control is enabled, edit the ACL file. Explicitly grant users or groups access to the `ADMIN` connection service.
- **LDAP user repository.** If you are using an LDAP user repository, do all of the following:
 - Enable access control.
 - Edit the ACL file and provide the names of users or groups who can make `ADMIN` connections. Specify any users or groups that are defined in the LDAP directory server.

Access Control for Physical Destinations

The destination access control section of the access control properties file contains physical destination-based access control rules. These rules determine who (users/groups) may do what (operations) where (physical destinations). The types of access that are regulated by these rules include sending messages to a queue, publishing messages to a topic, receiving messages from a queue, subscribing to a topic, and browsing messages in a queue.

By default, any user or group can have all types of access to any physical destination. You can add more specific destination access rules or edit the default rules. The rest of this section explains the syntax of physical destination access rules, which you must understand to write your own rules.

The syntax of destination rules is as follows:

resourceType.resourceVariant.operation.access.principalType=principals

Table 7–4 describes these elements:

TABLE 7–4 Elements of Physical Destination Access Control Rules

Component	Description
<i>resourceType</i>	Can be queue or topic.
<i>resourceVariant</i>	A physical destination name or all physical destinations (*), meaning all queues or all topics.
<i>operation</i>	Can be produce, consume, or browse.
<i>access</i>	Can be allow or deny.
<i>principalType</i>	Can be user or group.

Access can be given to one or more users and/or one or more groups.

The following examples illustrate different kinds of physical destination access control rules:

- Allow all users to send messages to any queue destinations.

```
queue.*.produce.allow.user=*
```

- Deny any member of the group user the ability to subscribe to the topic Admissions.

```
topic.Admissions.consume.deny.group=user
```

Access Control for Auto-Created Physical Destinations

The final section of the ACL properties file, includes access rules that specify for which users and groups the broker will auto-create a physical destination.

When a user creates a producer or consumer at a physical destination that does not already exist, the broker will create the destination if the broker’s auto-create property has been enabled.

By default, any user or group has the privilege of having a physical destination auto-created by the broker. This privilege is specified by the following rules:

```
queue.create.allow.user=*
topic.create.allow.user=*
```

You can edit the ACL file to restrict this type of access.

The general syntax for physical destination auto-create access rules is as follows:

resourceType.create.access.principalType=principals

Where *resourceType* is either queue or topic.

For example, the following rules allow the broker to auto-create topic destinations for everyone except Snoopy.

```
topic.create.allow.user=*
topic.create.deny.user=Snoopy
```

Note that the effect of physical destination auto-create rules must be congruent with that of physical destination access rules. For example, if you 1) change the destination access rule to forbid any user from sending a message to a destination but 2) enable the auto-creation of the destination, the broker *will* create the physical destination if it does not exist but it will *not* deliver a message to it.

Message Encryption

This section explains how to set up a connection service based on the Secure Socket Layer (SSL) standard, which sends encrypted messages between clients and broker. Message Queue supports the following SSL-based connection services:

- The `ssljms` service delivers secure, encrypted messages between a client and a broker, using the TCP/IP transport protocol.
- The `ssladmin` service creates a secure, encrypted connection between the Message Queue Command utility (`imqcmd`) and a broker, using the TCP/IP transport protocol. Encrypted connections are not supported for the Administration Console (`imqadmin`).
- The `cluster` service provides secure, encrypted communication between brokers in a cluster, using the TCP/IP transport protocol.
- The `httpsjms` service delivers secure, encrypted messages between a client and a broker, using an HTTPS tunnel servlet with the HTTP transport protocol.

The remainder of this section describes how to set up secure connections over TCP/IP, using the `ssljms`, `ssladmin`, and `cluster` connection services. For information on setting up secure connections over HTTP with the `httpsjms` service, see [Appendix C](#).

Using Self-Signed Certificates

To use an SSL-based connection service over TCP/IP, you generate a public/private 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. This section describes how to set up an SSL-based service using such self-signed certificates.

For a stronger level of authentication, you can use signed certificates verified by a certification authority. The use of signed certificates involves some additional steps beyond those needed for self-signed certificates: you must first perform the steps described in this section and then follow them with the additional ones in [“Using Signed Certificates” on page 146](#).

Message Queue's support for SSL with self-signed certificates is oriented toward securing on-the-wire data, on the assumption that the client is communicating with a known and trusted server. The following procedure shows the steps needed to set up an SSL-based connection service to use self-signed certificates. The subsections that follow describe each of these steps in greater detail.

▼ To Set Up an SSL-Based Connection Service Using Self-Signed Certificates

- 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.**

This step applies only to the `ssljms` connection service and not to `ssladmin` or `cluster`.

Generating a Self-Signed Certificate

Run the Key Tool utility (`imqkeytool`) to generate a self-signed certificate for the broker. (On UNIX® systems, you may need to run the utility as the superuser (`root`) in order to have permission to create the key store.) The same certificate can be used for the `ssljms`, `ssladmin`, or `cluster` connection service.

Enter the following at the command prompt:

```
imqkeytool -broker
```

The Key Tool utility prompts you for a key store password:

```
Generating keystore for the broker ...
Enter keystore password:
```

Next, the utility prompts you for information identifying the broker to which this certificate belongs. The information you supply will make up an X.500 distinguished name. [Table 7-5](#) shows the prompts and the values to be provided for each. Values are case-insensitive and can include spaces.

TABLE 7-5 Distinguished Name Information Required for a Self-Signed Certificate

Prompt	X.500 Attribute	Description	Example
What is your first and last name?	commonName (CN)	Fully qualified name of server running the broker	mqserver.sun.com
What is the name of your organizational unit?	organizationalUnit (OU)	Name of department or division	purchasing
What is the name of your organization?	organizationName (ON)	Name of larger organization, such as a company or government entity	My Company, Inc.
What is the name of your city or locality?	localityName (L)	Name of city or locality	San Francisco
What is the name of your state or province?	stateName (ST)	Full (unabbreviated) name of state or province	California
What is the two-letter country code for this unit?	country (C)	Standard two-letter country code	US

When you have entered the information, the Key Tool utility displays it for confirmation. For example:

```
Is CN=mqserver.sun.com, OU=purchasing, ON=My Company, Inc.,
L=San Francisco, ST=California, C=US correct?
```

To accept the current values and proceed, enter yes; to reenter values, accept the default or enter no. After you confirm, the utility pauses while it generates a key pair.

Next, the utility asks for a password to lock the key pair (key password). Press Return in response to this prompt to use the same password as the key password and key store password.

Note – Remember the password you specify. You must provide this password when you start the broker, to allow the broker to open the key store. You can store the key store password in a password file (see [“Password Files” on page 149](#)).

The Key Tool utility generates a self-signed certificate and places it in Message Queue’s key store. The key store is located in a directory that depends upon the operating system, as shown in [Appendix A](#).

The following are the configurable properties for the Message Queue key store for SSL-based connection services:

- `imq.keystore.file.dirpath`: The path to the directory containing the key store file. For the default value, see [Appendix A](#).
- `imq.keystore.file.name`: The name of the key store file.

- `imq.keystore.password`: The key store password.

In some circumstances, you may need to regenerate a key pair in order to solve certain problems: for example, if you forget the key store password or if 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 if you provided a key password different from the key store password when you generated the self-signed certificate.)

▼ To Regenerate a Key Pair

- 1 Remove the broker's key store, located as shown in [Appendix A](#).
- 2 Run `imqkeytool` again to generate a new key pair, as described above.

Enabling an SSL-Based Connection Service

To enable an SSL-based connection service in the broker, you need to add `ssljms` (or `ssladmin`) to the `imq.service.activelist` property.

▼ 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](#)):
`.../instances/instanceName/props/config.properties`
- 2 **Add an entry (if one does not already exist) for the `imq.service.activelist` property and include the desired SSL-based service(s) in the list.**
By default, the property includes the `jms` and `admin` connection services. Add the SSL-based service or services you wish to activate (`ssljms`, `ssladmin`, or both):
`imq.service.activelist=jms,admin,ssljms,ssladmin`

Note – The SSL-based cluster connection service is enabled using the `imq.cluster.transport` property rather than the `imq.service.activelist` property; see [“Connecting Brokers” on page 173](#).

- 3 **Save and close the instance configuration file.**

Starting the Broker

Start the broker, providing the key store password. You can provide the password in either of two ways:

- Allow the broker to prompt you for the password when it starts up:

```
imqbrokerd
Please enter Keystore password:
```

- Put the password in a password file, as described in [“Password Files” on page 149](#). Then set the property `imq.passfile.enabled = true` and do one of the following:

- Pass the location of the password file to the `imqbrokerd` command:

```
imqbrokerd -passfile /passfileDirectory/passfileName
```

- Start the broker without the `-passfile` option, but specify the location of the password file using the following two broker configuration properties:

```
imq.passfile.dirpath=/passfileDirectoryimq.passfile.name=passfileName
```

Note – When you start a broker or client with SSL, you may notice a sharp increase in CPU usage for a few seconds. This is because the JSSE (Java Secure Socket Extension) method `java.security.SecureRandom`, which Message Queue uses to generate random numbers, takes a significant amount of time to create the initial random number seed. Once the seed is created, the CPU usage level will drop to normal.

Configuring and Running an SSL-Based Client

The procedure for configuring a client to use an SSL-based connection service differs depending on whether it is an application client (using the `ssljms` connection service) or a Message Queue administrative client such as `imqcmd` (using the `ssladmin` connection service.)

Application Clients

For application clients, you must make sure the client has the following `.jar` files specified in its `CLASSPATH` variable:

```
imq.jar
jms.jar
```

If you are using a version of the Java 2 Software Development Kit (J2SDK) earlier than 1.4, you must also include the following Java Secure Socket Extension (JSSE) and Java Naming and Directory Interface (JNDI) `.jar` files:

```
jsse.jar
jnet.jar
jcert.jar
```

jndi.jar

(It is not necessary to include these files if you are using J2SDK 1.4 or later, which has JSSE and JNDI support built in.)

Once the CLASSPATH files are properly specified, one way to start the client and connect to the broker's ssljms connection service is by entering a command like the following:

```
java -DmqConnectionType=TLS clientAppName
```

This tells the connection to use an SSL-based connection service.

Administrative Clients

For administrative clients, you can establish a secure connection by including the `-secure` option when you invoke the `imqcmd` command: for example,

```
imqcmd list svc -b hostName:portNumber -u adminName -secure
```

where *adminName* is a valid entry in the Message Queue user repository. The command will prompt you for the password. (If you are using a flat-file repository, see [“Changing the Default Administrator Password” on page 132](#)).

Listing the connection services is a way to verify that the `ssladmin` service is running and that you can successfully make a secure administrative 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.

Using Signed Certificates

Signed certificates provide a stronger level of server authentication than self-signed certificates. You can implement signed certificates only between a client and broker, and not between

multiple brokers in a cluster. This requires the following extra steps in addition to the ones described above for configuring self-signed certificates. These steps are described in greater detail in the subsections that follow.

▼ To Use a Signed Certificate

- 1 **Install the certificate in the key store.**
- 2 **Configure the Message Queue client to require signed certificates when establishing an SSL-based connection to the broker.**

Obtaining and Installing a Signed Certificate

The following procedures explain how to obtain and install a signed certificate.

▼ To Obtain a Signed Certificate

- 1 **Use the J2SE `keytool` command to generate a certificate signing request (CSR) for the self-signed certificate you generated in the preceding section.**

Information about the `keytool` command can be found at

<http://java.sun.com/j2se/1.5.0/docs/tooldocs/solaris/keytool.html>

Here is an example:

```
keytool -certreq -keyalg RSA -alias imq -file certreq.csr
        -keystore /etc/imq/keystore -storepass myStorePassword
```

This generates a CSR encapsulating the certificate in the specified file (`certreq.csr` in the example).

- 2 **Use the CSR to generate or request a signed certificate.**

You can do this by either of the following methods:

- Have the certificate signed by a well known certification authority (CA), such as Thawte or Verisign. See your CA's documentation for more information on how to do this.
- Sign the certificate yourself, 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 may arrive as an e-mail attachment or in the text of a message.

- 3 **Save the signed certificate in a file.**

The instructions below use the example name `broker.cer` to represent the broker certificate.

▼ To Install a Signed Certificate

1 Check whether J2SE supports your certification authority by default.

The following command lists the root CAs in the system key store:

```
keytool -v -list -keystore $JAVA_HOME/lib/security/cacerts
```

If your CA is listed, skip the next step.

2 If your certification authority is not supported in J2SE, import the CA's root certificate into the Message Queue key store.

Here is an example:

```
keytool -import -alias ca -file ca.cer -noprompt -trustcacerts  
-keystore /etc/imq/keystore -storepass myStorePassword
```

where `ca.cer` is the file containing the 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.

3 Import the signed certificate into the key store to replace the original self-signed certificate.

Here is an example:

```
keytool -import -alias imq -file broker.cer -noprompt -trustcacerts  
-keystore /etc/imq/keystore -storepass myStorePassword
```

where `broker.cer` is the file containing the signed certificate that you received from the CA.

The Message Queue key store now contains a signed certificate to use for SSL connections.

Configuring the Message Queue Client Runtime to Require Signed Certificates

You must now configure the Message Queue client runtime to require signed certificates, and ensure that it trusts the certification authority that signed the certificate.

▼ To Configure the Client Runtime to Require Signed Certificates

1 Set the connection factory's `imqSSLIsHostTrusted` attribute to `false`.

By default, the `imqSSLIsHostTrusted` attribute of the connection factory object that the client will be using to establish broker connections is set to `true`, meaning that the client runtime will accept any certificate presented to it. You must change this value to `false` so that the client runtime will attempt to validate all certificates presented to it. Validation will fail if the signer of the certificate is not in the client's trust store.

2 Verify whether the signing authority is registered in the client's trust store.

To test whether the client will accept certificates signed by your certification authority, try to establish an SSL connection, as described above under “[Configuring and Running an SSL-Based Client](#)” on page 145. If the CA is in the client's trust store, the connection will succeed and you can skip the next step. If the connection fails with a certificate validation error, go on to the next step.

3 Install the signing CA's root certificate in the client's trust store.

The client searches the key store files `cacerts` and `jssecacerts` by default, so no further configuration is necessary if you install the certificate in either of those files. The following example installs a test root certificate from the Verisign certification authority from a file named `testrootca.cer` into the default system certificate file, `cacerts`. The example assumes that J2SE is installed in the directory `$JAVA_HOME/usr/j2se`:

```
keytool -import -keystore /usr/j2se/jre/lib/security/cacerts
        -alias VerisignTestCA -file testrootca.cer -noprompt
        -trustcacerts -storepass myStorePassword
```

An alternative (and recommended) option is to install the root certificate into the alternative system certificate file, `jssecacerts`:

```
keytool -import -keystore /usr/j2se/jre/lib/security/jssecacerts
        -alias VerisignTestCA -file testrootca.cer -noprompt
        -trustcacerts -storepass myStorePassword
```

A third possibility is to install the root certificate into some other key store file and configure the client to use that as its trust store. The following example installs into the file `/home/smith/.keystore`:

```
keytool -import -keystore /home/smith/.keystore
        -alias VerisignTestCA -file testrootca.cer -noprompt
        -trustcacerts -storepass myStorePassword
```

Since the client does not search this key store by default, you must explicitly provide its location to the client to use as a trust store. You do this by setting the Java system property `javax.net.ssl.trustStore` once the client is running:

```
javax.net.ssl.trustStore=/home/smith/.keystore
```

Password Files

Several types of commands require passwords. In [Table 7–6](#), the first column lists the commands that require passwords and the second column lists the reason that passwords are needed.

TABLE 7-6 Commands That Use Passwords

Command	Purpose	Purpose of Password
imqbrokerd	Start the broker	Access a JDBC-based persistent data store, an SSL certificate key store, or an LDAP user repository
imqcmd	Manage the broker	Authenticate an administrative user who is authorized to use the command
imqdbmgr	Manage a JDBC-based data store	Access the data store

You can specify these passwords in a *password file* and use the `-passfile` option to specify the name of the file. This is the format for the `-passfile` option:

```
imqbrokerd -passfile myPassfile
```

Note – In previous releases, you could use the `-p`, `-password`, `-dbpassword`, and `-ldappassword` options to specify passwords on a command line. These options are deprecated and will be removed in a future release. In the current release, a value on the command line for one of these options supersedes the associated value in a password file.

Security Concerns

Specifying a password interactively, in response to a prompt, is the most secure method of specifying a password, unless your monitor is visible to other people. You can also specify a password file on the command line. For non-interactive use of commands, however, you must use a password file.

A password file is unencrypted, so you must set its permissions to protect it from unauthorized access. Set permissions such that they limit the users who can view the file, but provide read access to the user who starts the broker.

Password File Contents

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

A password file can contain the passwords shown in [Table 7-7](#):

TABLE 7-7 Passwords in a Password File

Password	Affected Commands	Description
imq.imqcmd.password	+imqcmd	Specifies the administrator password for an imqcmd command line. The password is authenticated for each command.
imq.keystore.password	imqbrokerd	Specifies the key store password for SSL-based services.
imq.persist.jdbc.password	imqbrokerd imdbmgr	Specifies the password used to open a database connection, if required.
imq.user_repository.ldap.password	imqbrokerd	Specifies the password associated with the distinguished name assigned to a broker for binding to a configured LDAP user repository.

A sample password file is part of the Message Queue product. For the location of the sample file, see [Appendix A](#)

Connecting Through a Firewall

When a client application is separated from the broker by a firewall, special measures are needed in order to establish a connection. One approach is to use the `httpjms` or `httpsjms` connection service, which can “tunnel” through the firewall; see [Appendix C](#) for details. HTTP connections are slower than other connection services, however; a faster alternative is to bypass the Message Queue Port Mapper and explicitly assign a static port address to the desired connection service, and then open that specific port in the firewall. This approach can be used to connect through a firewall using the `jms` or `ssljms` connection service (or, in unusual cases, `admin` or `ssladmin`).

TABLE 7-8 Broker Configuration Properties for Static Port Addresses

Connection Service	Configuration Property
jms	imq.jms.tcp.port
ssljms	imq.ssljms.tls.port
admin	imq.admin.tcp.port
ssladmin	imq.ssladmin.tls.port

▼ To Enable Broker Connections Through a Firewall

1 Assign a static port address to the connection service you wish to use.

To bypass the Port Mapper and assign a static port number directly to a connection service, set the broker configuration property `imq.serviceName.protocolType.port`, where *serviceName* is the name of the connection service and *protocolType* is its protocol type (see [Table 7–8](#)). As with all broker configuration properties, you can specify this property either in the broker's instance configuration file or from the command line when starting the broker. For example, to assign port number 10234 to the `jms` connection service, either include the line

```
imq.jms.tcp.port=10234
```

in the configuration file or start the broker with the command

```
imqbrokerd -name myBroker -Dimq.jms.tcp.port=10234
```

2 Configure the firewall to allow connections to the port number you assigned to the connection service.

You must also allow connections through the firewall to Message Queue's Port Mapper port (normally 7676, unless you have reassigned the Port Mapper to some other port). In the example above, for instance, you would need to open the firewall for ports 10234 and 7676.

Audit Logging

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`.

Managing Administered Objects

Administered objects encapsulate provider-specific configuration and naming information, enabling the development of client applications that are portable from one JMS provider to another. A Message Queue™ administrator typically creates administered objects for client applications to use in obtaining broker connections for sending and receiving messages.

This chapter tells how to use the Object Manager utility (`imqobjmgr`) to create and manage administered objects. It contains the following sections:

- “Object Stores” on page 153
- “Administered Object Attributes” on page 156
- “Using the Object Manager Utility” on page 163
- “Adding Administered Objects” on page 164
- “Listing Administered Objects” on page 166
- “Viewing Administered Object Information” on page 166
- “Modifying Administered Object Attributes” on page 167

Object Stores

Administered objects are placed in a readily available object store where they can be accessed by client applications via the Java Naming and Directory Interface (JNDI). There are two types of object store you can use: a standard Lightweight Directory Access Protocol (LDAP) directory server or a directory in the local file system.

LDAP Server Object Stores

An LDAP server is the recommended object store for production messaging systems. LDAP servers are designed for use in distributed systems and provide security features that are useful in production environments.

LDAP implementations are available from a number of vendors. To manage an object store on an LDAP server with Message Queue administration tools, you may first need to configure the server to store Java objects and perform JNDI lookups; see the documentation provided with your LDAP implementation for details.

To use an LDAP server as your object store, you must specify the attributes shown in [Table 8–1](#). These attributes fall into the following categories:

- **Initial context.** The `java.naming.factory.initial` attribute specifies the initial context for JNDI lookups on the server. The value of this attribute is fixed for a given LDAP object store.
- **Location.** The `java.naming.provider.url` attribute specifies the URL and directory path for the LDAP server. You must verify that the specified directory path exists.
- **Security.** The attributes `java.naming.security.principal`, `java.naming.security.credentials`, and `java.naming.security.authentication` govern the authentication of callers attempting to access the object store. The exact format and values of these attributes depend on the LDAP service provider; see the documentation provided with your LDAP implementation for details and to determine whether security information is required on all operations or only on those that change the stored data.

TABLE 8–1 LDAP Object Store Attributes

Attribute	Description
<code>java.naming.factory.initial</code>	Initial context for JNDI lookup Example: <code>com.sun.jndi.ldap.LdapCtxFactory</code>
<code>java.naming.provider.url</code>	Server URL and directory path Example: <code>ldap://myD.com:389/ou=mq1,o=App</code> where administered objects are stored in the directory <code>/App/mq1</code> .
<code>java.naming.security.principal</code>	Identity of the principal for authenticating callers The format of this attribute depends on the authentication scheme: for example, <code>uid=homerSimpson,ou=People,o=mq</code> If this attribute is unspecified, the behavior is determined by the LDAP service provider.

TABLE 8-1 LDAP Object Store Attributes (Continued)

Attribute	Description
<code>java.naming.security.credentials</code>	<p>Credentials of the authentication principal</p> <p>The value of this attribute depends on the authentication scheme: for example, it might be a hashed password, a clear-text password, a key, or a certificate.</p> <p>If this property is unspecified, the behavior is determined by the LDAP service provider.</p>
<code>java.naming.security.authentication</code>	<p>Security level for authentication</p> <p>The value of this attribute is one of the keywords <code>none</code>, <code>simple</code>, or <code>strong</code>. For example, If you specify <code>simple</code>, you will be prompted for any missing principal or credential values. This will allow you a more secure way of providing identifying information.</p> <p>If this property is unspecified, the behavior is determined by the LDAP service provider.</p>

File-System Object Stores

Message Queue also supports the use of a directory in the local file system as an object store for administered objects. While this approach is not recommended for production systems, it has the advantage of being very easy to use in development environments. Note, however, that for a directory to be used as a centralized object store for clients deployed across multiple computer nodes, all of those clients must have access to the directory. In addition, any user with access to the directory can use Message Queue administration tools to create and manage administered objects.

To use a file-system directory as your object store, you must specify the attributes shown in [Table 8-2](#). These attributes have the same general meanings described above for LDAP object stores; in particular, the `java.naming.provider.url` attribute specifies the directory path of the directory holding the object store. This directory must exist and have the proper access permissions for the user of Message Queue administration tools as well as the users of the client applications that will access the store.

TABLE 8-2 File-system Object Store Attributes

Attribute	Description
<code>java.naming.factory.initial</code>	<p>Initial context for JNDI lookup</p> <p>Example:</p> <p><code>com.sun.jndi.fscontext.RefFSContextFactory</code></p>

TABLE 8-2 File-system Object Store Attributes (Continued)

Attribute	Description
java.naming.provider.url	Directory path Example: file:///C:/myapp/mqobjs

Administered Object Attributes

Message Queue administered objects are of two basic kinds:

- *Connection factories* are used by client applications to create connections to brokers.
- *Destinations* represent locations on a broker with which client applications can exchange (send and retrieve) messages.

Note – A special SOAP *endpoint administered object* is used for SOAP messaging; see the *Message Queue Developer's Guide for Java Clients* for more information.

Each type of administered object has certain attributes that determine the object's properties and behavior. This section describes how to use the Object Manager command line utility (`imqobjmgr`) to set these attributes; you can also set them with the GUI Administration Console, as described in [“Working With Administered Objects” on page 56](#).

Connection Factory Attributes

Client applications use *connection factory* administered objects to create connections with which to exchange messages with a broker. A connection factory's attributes define the properties of all connections it creates. Once a connection has been created, its properties cannot be changed; thus the only way to configure a connection's properties is by setting the attributes of the connection factory used to create it.

Message Queue defines two classes of connection factory objects:

- `ConnectionFactory` objects support normal messaging and nondistributed transactions.
- `XAConnectionFactory` objects support distributed transactions.

Both classes share the same configuration attributes, which you can use to optimize resources, performance, and message throughput. These attributes are listed and described in detail in [Chapter 16](#) and are discussed in the following sections below:

- [“Connection Handling” on page 157](#)
- [“Client Identification” on page 159](#)

- “Reliability And Flow Control” on page 161
- “Queue Browser and Server Sessions” on page 161
- “Standard Message Properties” on page 162
- “Message Header Overrides” on page 162

Connection Handling

Connection handling attributes specify the broker address to which to connect and, if required, how to detect connection failure and attempt reconnection. They are summarized in [Table 16–1](#).

Broker Address List

The most important connection handling attribute is `imqAddressList`, which specifies the broker or brokers to which to establish a connection. The value of this attribute is a string containing a broker address or (in the case of a broker cluster) multiple addresses separated by commas. Broker addresses can use a variety of addressing schemes, depending on the connection service to be used (see “[Connection Services](#)” on page 76) and the method of establishing a connection:

- `mq` uses the broker’s Port Mapper to assign a port dynamically for either the `jms` or `ssljms` connection service.
- `mqtcp` bypasses the Port Mapper and connects directly to a specified port, using the `jms` connection service.
- `mqssl` makes a Secure Socket Layer (SSL) connection to a specified port, using the `ssljms` connection service.
- `http` makes a Hypertext Transport Protocol (HTTP) connection to a Message Queue tunnel servlet at a specified URL, using the `httpjms` connection service.
- `https` makes a Secure Hypertext Transport Protocol (HTTPS) connection to a Message Queue tunnel servlet at a specified URL, using the `httpsjms` connection service.

These addressing schemes are summarized in [Table 16–2](#).

The general format for each broker address is

scheme://address

where *scheme* is one of the addressing schemes listed above and *address* denotes the broker address itself. The exact syntax for specifying the address varies depending on the addressing scheme, as shown in the last column of [Table 16–2](#). [Table 16–3](#) shows examples of the various address formats.

In a multiple-broker cluster environment, the address list can contain more than one broker address. If the first connection attempt fails, the Message Queue client runtime will attempt to connect to another address in the list, and so on until the list is exhausted. Two additional connection factory attributes control the way this is done:

- `imqAddressListBehavior` specifies the order in which to try the specified addresses. If this attribute is set to the string `PRIORITY`, addresses will be tried in the order in which they appear in the address list. If the attribute value is `RANDOM`, the addresses will instead be tried in random order; this is useful, for instance, when many Message Queue clients are sharing the same connection factory object, to prevent them from all attempting to connect to the same broker address.
- `imqAddressListIterations` specifies how many times to cycle through the list before giving up and reporting failure. A value of `-1` denotes an unlimited number of iterations: the client runtime will keep trying until it succeeds in establishing a connection or until the end of time, whichever occurs first.

Automatic Reconnection

By setting a connection factory's `imqReconnectEnabled` attribute to `true`, you can enable a client to reconnect automatically to a broker if a connection fails. The `imqReconnectAttempts` attribute controls the number of reconnection attempts to a given broker address; `imqReconnectInterval` specifies the interval, in milliseconds, to wait between attempts.

In a broker cluster, where the broker address list (`imqAddressList`) specifies multiple addresses, a failed connection can be restored not only on the original broker, but also on a different one in the cluster. If reconnection to the original broker fails, the client runtime will try the other addresses in the list. The `imqAddressListBehavior` and `imqAddressListIterations` attributes control the order in which addresses are tried and the number of iterations through the list, as described in the preceding section. Each address is tried repeatedly at intervals of `imqReconnectInterval` milliseconds, up to the maximum number of attempts specified by `imqReconnectAttempts`.

Automatic reconnection supports all client acknowledgment modes for message consumption. Once a connection has been reestablished, the broker will redeliver all unacknowledged messages it had previously delivered, marking them with a `Redeliver` flag. Application code can use this flag to determine whether any message has already been consumed but not yet acknowledged. (In the case of nondurable subscribers, however, the broker does not hold messages once their connections have been closed. Thus any messages produced for such subscribers while the connection is down cannot be delivered after reconnection and will be lost.) Message production is blocked while automatic reconnection is in progress; message producers cannot send messages to the broker until after the connection has been reestablished.

Automatic reconnection provides connection failover, but not data failover: persistent messages and other state information held by a failed or disconnected broker can be lost when the client is reconnected to a different broker instance. While attempting to reestablish a connection, Message Queue does maintain objects (such as sessions, message consumers, and message producers) provided by the client runtime. Temporary destinations are also maintained for a time when a connection fails, because clients might reconnect and access them again; after giving clients time to reconnect and use these destinations, the broker will delete them. In circumstances where the client-side state cannot be fully restored on the broker on

reconnection (for example, when using transacted sessions, which exist only for the duration of a connection), automatic reconnection will not take place and the connection's exception handler will be called instead. It is then up to the application code to catch the exception, reconnect, and restore state.

Periodic Testing (Pinging) of Connections

The Message Queue client runtime can be configured to periodically test, or “ping,” a connection, allowing connection failures to be detected preemptively before an attempted message transmission fails. Such testing is particularly important for client applications that only consume messages and do not produce them, since such applications cannot otherwise detect when a connection has failed. Clients that produce messages only infrequently can also benefit from this feature.

The connection factory attribute `imqPingInterval` specifies the frequency, in seconds, with which to ping a connection. By default, this interval is set to 30 seconds; a value of -1 disables the ping operation.

The response to an unsuccessful ping varies from one operating-system platform to another. On some operating systems, an exception is immediately thrown to the client application's exception listener. (If the client does not have an exception listener, its next attempt to use the connection will fail.) Other systems may continue trying to establish a connection to the broker, buffering successive pings until one succeeds or the buffer overflows.

Client Identification

The connection factory attributes listed in [Table 16–4](#) support client authentication and the setting of client identifiers for durable subscribers.

Client Authentication

All attempts to connect to a broker must be authenticated by user name and password against a user repository maintained by the message service. The connection factory attributes `imqDefaultUsername` and `imqDefaultPassword` specify a default user name and password to be used if the client does not supply them explicitly when creating a connection.

As a convenience for developers who do not wish to bother populating a user repository during application development and testing, Message Queue provides a guest user account with user name and password both equal to `guest`. This is also the default value for the `imqDefaultUsername` and `imqDefaultPassword` attributes, so that if they are not specified explicitly, clients can always obtain a connection under the guest account. In a production environment, access to broker connections should be restricted to users who are explicitly registered in the user repository.

Client Identifier

The *Java Message Service Specification* requires that a connection provide a unique *client identifier* whenever the broker must maintain a persistent state on behalf of a client. Message Queue uses such client identifiers to keep track of durable subscribers to a topic destination. When a durable subscriber becomes inactive, the broker retains all incoming messages for the topic and delivers them when the subscriber becomes active again. The broker identifies the subscriber by means of its client identifier.

While it is possible for a client application to set its own client identifier programmatically using the connection object's `setClientID` method, this makes it difficult to coordinate client identifiers to ensure that each is unique. It is generally better to have Message Queue automatically assign a unique identifier when creating a connection on behalf of a client. This can be done by setting the connection factory's `imqConfiguredClientID` attribute to a value of the form

`${u}factoryID`

The characters `${u}` must be the first four characters of the attribute value. (Any character other than `u` between the braces will cause an exception to be thrown on connection creation; in any other position, these characters have no special meaning and will be treated as plain text.) The value for *factoryID* is a character string uniquely associated with this connection factory object.

When creating a connection for a particular client, Message Queue will construct a client identifier by replacing the characters `${u}` with `u:userName`, where *userName* is the user name authenticated for the connection. This ensures that connections created by a given connection factory, although identical in all other respects, will each have their own unique client identifier. For example, if the user name is Calvin and the string specified for the connection factory's `imqConfiguredClientID` attribute is `${u}Hobbes`, the client identifier assigned will be `u:CalvinHobbes`.

Note – This scheme will not work if two clients both attempt to obtain connections using the default user name `guest`, since each would have a client identifier with the same `${u}` component. In this case, only the first client to request a connection will get one; the second client's connection attempt will fail, because Message Queue cannot create two connections with the same client identifier.

Even if you specify a client identifier with `imqConfiguredClientID`, client applications can override this setting with the connection method `setClientID`. You can prevent this by setting the connection factory's `imqDisableSetClientID` attribute to `true`. Note that for an application that uses durable subscribers, the client identifier *must* be set one way or the other: either administratively with `imqConfiguredClientID` or programmatically with `setClientID`.

Reliability And Flow Control

Because “payload” messages sent and received by clients and control messages (such as broker acknowledgments) used by Message Queue itself pass over the same client-broker connection, excessive levels of payload traffic can interfere with the delivery of control messages. To help alleviate this problem, the connection factory attributes listed in [Table 16–5](#) allow you to manage the relative flow of the two types of message. These attributes fall into four categories:

- **Acknowledgment timeout** specifies the maximum time (`imqAckTimeout`) to wait for a broker acknowledgment before throwing an exception.
- **Connection flow metering** limits the transmission of payload messages to batches of a specified size (`imqConnectionFlowCount`), ensuring periodic opportunities to deliver any accumulated control messages.
- **Connection flow control** limits the number of payload messages (`imqConnectionFlowLimit`) that can be held pending on a connection, waiting to be consumed. When the limit is reached, delivery of payload messages to the connection is suspended until the number of messages awaiting consumption falls below the limit. Use of this feature is controlled by a boolean flag (`imqConnectionFlowLimitEnabled`).
- **Consumer flow control** limits the number of payload messages (`imqConsumerFlowLimit`) that can be held pending for any single consumer, waiting to be consumed. (This limit can also be specified as a property of a specific queue destination, `consumerFlowLimit`.) When the limit is reached, delivery of payload messages to the consumer is suspended until the number of messages awaiting consumption, as a percentage of `imqConsumerFlowLimit`, falls below the limit specified by the `imqConsumerFlowThreshold` attribute. This helps improve load balancing among multiple consumers by preventing any one consumer from starving others on the same connection.

The use of any of these flow control techniques involves a tradeoff between reliability and throughput; see “[Client Runtime Message Flow Adjustments](#)” on page 213 for further discussion.

Queue Browser and Server Sessions

[Table 16–6](#) lists connection factory attributes affecting client queue browsing and server sessions. The `imqQueueBrowserMaxMessagesPerRetrieve` attribute specifies the maximum number of messages to retrieve at one time when browsing the contents of a queue destination; `imqQueueBrowserRetrieveTimeout` gives the maximum waiting time for retrieving them. (Note that `imqQueueBrowserMaxMessagesPerRetrieve` does not affect the total number of messages browsed, only the way they are chunked for delivery to the client runtime: fewer but larger chunks or more but smaller ones. The client application will always receive all messages in the queue. Changing the attribute’s value may affect performance, but will not affect the total amount of data retrieved.) The boolean attribute `imqLoadMaxToServerSession` governs the behavior of connection consumers in an application server session: if the value of this attribute is `true`, the client will load up to the maximum number of messages into a server session; if `false`, it will load only a single message at a time.

Standard Message Properties

The *Java Message Service Specification* defines certain standard message properties, which JMS providers (such as Message Queue) may optionally choose to support. By convention, the names of all such standard properties begin with the letters JMSX. The connection factory attributes listed in [Table 16–7](#) control whether the Message Queue client runtime sets certain of these standard properties. For produced messages, these include the following properties:

- `JMSXUserID` Identity of the user sending the message
- `JMSXAppID` Identity of the application sending the message
- `JMSXProducerTXID` Transaction identifier of the transaction within which the message was produced

For consumed messages, they include

- `JMSXConsumerTXID` Transaction identifier of the transaction within which the message was consumed
- `JMSXRcvTimeStamp` Time the message was delivered to the consumer

Message Header Overrides

You can use the connection factory attributes listed in [Table 16–8](#) to override the values set by a client for certain JMS message header fields. The settings you specify will be used for all messages produced by connections obtained from that connection factory. Header fields that you can override in this way are

- `JMSDeliveryMode` Delivery mode (persistent or nonpersistent)
- `JMSExpiration` Expiration time
- `JMSPriority` Priority level

There are two attributes for each of these fields: one boolean, to control whether the field can be overridden, and another to specify its value. For instance, the attributes for setting the priority level are `imqOverrideJMSPriority` and `imqJMSPriority`. There is also an additional attribute, `imqOverrideJMSHeadersToTemporaryDestinations`, that controls whether override values apply to temporary destinations.

Note – Because overriding message headers may interfere with the needs of specific applications, these attributes should only be used in consultation with an application’s designers or users.

Destination Attributes

The *destination* administered object that identifies a physical queue or topic destination has only two attributes, listed in [Table 16–9](#). The important one is `imqDestinationName`, which gives the name of the physical destination that this administered object represents; this is the

name that was specified with the `-n` option to the `imqcmd create dst` command that created the physical destination. (Note that there is not necessarily a one-to-one relationship between destination administered objects and the physical destinations they represent: a single physical destination can be referenced by more than one administered object, or by none at all.) There is also an optional descriptive string, `imqDestinationDescription`, which you can use to help identify the destination object and distinguish it from others you may have created.

Using the Object Manager Utility

The Message Queue Object Manager utility (`imqobjmgr`) allows you to create and manage administered objects. The `imqobjmgr` command provides the following subcommands for performing various operations on administered objects:

<code>add</code>	Add an administered object to an object store
<code>delete</code>	Delete an administered object from an object store
<code>list</code>	List existing administered objects in an object store
<code>query</code>	Display information about an administered object
<code>update</code>	Modify the attributes of an administered object

See “[Object Manager Utility](#)” on page 256 for reference information about the syntax, subcommands, and options of the `imqobjmgr` command.

Most Object Manager operations require you to specify the following information as options to the `imqobjmgr` command:

- The **JNDI lookup name** (`-l`) of the administered object
This is the logical name by which client applications can look up the administered object in the object store, using the Java Naming and Directory Interface.
- The **attributes of the JNDI object store** (`-j`)
See “[Object Stores](#)” on page 153 for information on the possible attributes and their values.
- The **type** (`-t`) of the administered object
Possible types include the following:

<code>q</code>	Queue destination
<code>t</code>	Topic destination
<code>cf</code>	Connection factory
<code>qf</code>	Queue connection factory
<code>tf</code>	Topic connection factory
<code>xcf</code>	Connection factory for distributed transactions

- xqf Queue connection factory for distributed transactions
- xtf Topic connection factory for distributed transactions
- e SOAP endpoint
- The **attributes** (-o) of the administered object
See “[Administered Object Attributes](#)” on page 156 for information on the possible attributes and their values.

Adding Administered Objects

The `imqobjmgr` command's `add` subcommand adds administered objects for connection factories and topic or queue destinations to the object store. Administered objects stored in an LDAP object store must have lookup names beginning with the prefix `cn=`; lookup names in a file-system object store need not begin with any particular prefix, but must not include the slash character (/).

Note – The Object Manager lists and displays only Message Queue administered objects. If an object store should contain a non-Message Queue object with the same lookup name as an administered object that you wish to add, you will receive an error when you attempt the add operation.

Adding a Connection Factory

To enable client applications to create broker connections, add a connection factory administered object for the type of connection to be created: a queue connection factory or a topic connection factory. [Example 8–1](#) shows a command to add a queue connection factory (administered object type `qf`) to an LDAP object store. The object has lookup name `cn=myQCF` and connects to a broker running on host `myHost` at port number 7272, using the `jms` connection service.

EXAMPLE 8–1 Adding a Connection Factory

```
imqobjmgr add
-l "cn=myQCF"
-j "java.naming.factory.initial=com.sun.jndi.ldap.LdapCtxFactory"
-j "java.naming.provider.url=ldap://mydomain.com:389/o=imq"
-j "java.naming.security.principal=uid=homerSimpson,ou=People,o=imq"
-j "java.naming.security.credentials=doh"
-j "java.naming.security.authentication=simple"
-t qf
-o "imqAddressList=mq://myHost:7272/jms"
```

Adding a Destination

When creating an administered object representing a destination, it is good practice to create the physical destination first, before adding the administered object to the object store. Use the Command utility (`imqcmd`) to create the physical destination, as described in [“Creating a Physical Destination” on page 115](#).

The command shown in [Example 8–2](#) adds an administered object to an LDAP object store representing a topic destination with lookup name `myTopic` and physical destination name `physTopic`. The command for adding a queue destination would be similar, except that the administered object type (`-t` option) would be `q` (for “queue destination”) instead of `t` (for “topic destination”).

EXAMPLE 8–2 Adding a Destination to an LDAP Object Store

```
imqobjmgr add
-l "cn=myTopic"
-j "java.naming.factory.initial=com.sun.jndi.ldap.LdapCtxFactory"
-j "java.naming.provider.url=ldap://mydomain.com:389/o=imq"
-j "java.naming.security.principal=uid=homerSimpson,ou=People,o=imq"
-j "java.naming.security.credentials=doh"
-j "java.naming.security.authentication=simple"
-t t
-o "imqDestinationName=physTopic"
```

[Example 8–3](#) shows the same command, but with the administered object stored in a Solaris file system instead of an LDAP server.

EXAMPLE 8–3 Adding a Destination to a File-System Object Store

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

Deleting Administered Objects

To delete an administered object from the object store, you use the `delete` subcommand of the `imqobjmgr` command, specify lookup name, type, and location of the object to be deleted. The command shown in [Example 8–4](#) deletes the object that was added in [“Adding a Destination” on page 165](#) above.

EXAMPLE 8-4 Deleting an Administered Object

```
imqobjmgr delete
-l "cn=myTopic"
-j "java.naming.factory.initial=com.sun.jndi.ldap.LdapCtxFactory"
-j "java.naming.provider.url=ldap://mydomain.com:389/o=imq"
-j "java.naming.security.principal=uid=homerSimpson,ou=People,o=imq"
-j "java.naming.security.credentials=doh"
-j "java.naming.security.authentication=simple"
-t t
```

Listing Administered Objects

You can use the Object Manager's `list` subcommand to get a list of all administered objects in an object store or those of a specific type. [Example 8-5](#) shows how to list all administered objects on an LDAP server.

EXAMPLE 8-5 Listing All Administered Objects

```
imqobjmgr list
-j "java.naming.factory.initial=com.sun.jndi.ldap.LdapCtxFactory"
-j "java.naming.provider.url=ldap://mydomain.com:389/o=imq"
-j "java.naming.security.principal=uid=homerSimpson,ou=People,o=imq"
-j "java.naming.security.credentials=doh"
-j "java.naming.security.authentication=simple"
```

[Example 8-6](#) lists all queue destinations (type `q`).

EXAMPLE 8-6 Listing Administered Objects of a Specific Type

```
imqobjmgr list
-j "java.naming.factory.initial=com.sun.jndi.ldap.LdapCtxFactory"
-j "java.naming.provider.url=ldap://mydomain.com:389/o=imq"
-j "java.naming.security.principal=uid=homerSimpson,ou=People,o=imq"
-j "java.naming.security.credentials=doh"
-j "java.naming.security.authentication=simple"
-t q
```

Viewing Administered Object Information

The query subcommand displays information about a specified administered object, identified by its lookup name and the attributes of the object store containing it. [Example 8-7](#) displays information about an object whose lookup name is `cn=myTopic`.

EXAMPLE 8-7 Viewing Administered Object Information

```
imqobjmgr query
-l "cn=myTopic"
-j "java.naming.factory.initial=com.sun.jndi.ldap.LdapCtxFactory"
-j "java.naming.provider.url=ldap://mydomain.com:389/o=imq"
-j "java.naming.security.principal=uid=homerSimpson,ou=People,o=imq"
-j "java.naming.security.credentials=doh"
-j "java.naming.security.authentication=simple"
```

Modifying Administered Object Attributes

To modify the attributes of an administered object, use the `imqobjmgr update` subcommand. You supply the object's lookup name and location, and use the `-o` option to specify the new attribute values.

[“Modifying Administered Object Attributes” on page 167](#) changes the value of the `imqReconnectAttempts` attribute for the queue connection factory that was added to the object store in [Example 8-8](#).

EXAMPLE 8-8 Modifying an Administered Object's Attributes

```
imqobjmgr update
-l "cn=myQCF"
-j "java.naming.factory.initial=com.sun.jndi.ldap.LdapCtxFactory"
-j "java.naming.provider.url=ldap://mydomain.com:389/o=imq"
-j "java.naming.security.principal=uid=homerSimpson,ou=People,o=imq"
-j "java.naming.security.credentials=doh"
-j "java.naming.security.authentication=simple"
-t qf
-o "imqReconnectAttempts=3"
```

Using Command Files

The `-i` option to the `imqobjmgr` command allows you to specify the name of a command file that uses Java property file syntax to represent all or part of the subcommand clause. This feature is especially useful for specifying object store attributes, which typically require a lot of typing and are likely to be the same across multiple invocations of `imqobjmgr`. Using a command file can also allow you to avoid exceeding the maximum number of characters allowed for the command line.

[Example 8–9](#) shows the general syntax for an Object Manager command file. Note that the version property is not a command line option: it refers to the version of the command file itself (not that of the Message Queue product) and must be set to the value 2.0.

EXAMPLE 8–9 Object Manager Command File Syntax

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

As an example, consider the Object Manager command shown earlier in [Example 8–1](#), which adds a queue connection factory to an LDAP object store. This command can be encapsulated in a command file as shown in [Example 8–10](#). If the command file is named `MyCmdFile`, you can then execute the command with the command line

```
imqobjmgr -i MyCmdFile
```

EXAMPLE 8–10 Example Command File

```
version=2.0
cmdtype=add
obj.lookupName=cn=myQCF
objstore.attrs.java.naming.factory.initial=com.sun.jndi.ldap.LdapCtxFactory
objstore.attrs.java.naming.provider.url=ldap://mydomain.com:389/o=imq
objstore.attrs.java.naming.security.principal=\
                                uid=homerSimpson,ou=People,o=imq
objstore.attrs.java.naming.security.credentials=doh
objstore.attrs.java.naming.security.authentication=simple
obj.type=qf
obj.attrs.imqAddressList=mq://myHost:7272/jms
```

A command file can also be used to specify only part of the `imqobjmgr` subcommand clause, with the remainder supplied explicitly on the command line. For example, the command file shown in [Example 8–11](#) specifies only the attribute values for an LDAP object store.

EXAMPLE 8-11 Partial Command File

```

version=2.0
objstore.attrs.java.naming.factory.initial=com.sun.jndi.ldap.LdapCtxFactory
objstore.attrs.java.naming.provider.url=ldap://mydomain.com:389/o=imq
objstore.attrs.java.naming.security.principal=\\
                                     uid=homerSimpson,ou=People,o=imq
objstore.attrs.java.naming.security.credentials=doh
objstore.attrs.java.naming.security.authentication=simple

```

You could then use this command file to specify the object store in an `imqobjmgr` command while supplying the remaining options explicitly, as shown in [Example 8-12](#).

EXAMPLE 8-12 Using a Partial Command File

```

imqobjmgr add
-l "cn=myQCF"
-i MyCmdFile
-t qf
-o "imqAddressList=mq://myHost:7272/jms"

```

Additional examples of command files can be found at the following locations, depending on your platform:

Solaris: `/usr/demo/imq/imqobjmgr` **Linux:** `/opt/sun/mq/examples/imqobjmgr` **Windows:** `IMQ_HOME/demo/imqobjmgr`

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 service to scale its operations with the volume of message traffic by distributing client connections among multiple brokers. See the *Message Queue Technical Overview* for a general discussion of clusters and how they operate.

This chapter describes how to manage broker clusters, connect brokers to them, and configure them. It contains the following sections:

- “Cluster Configuration Properties” on page 171
- “Managing Clusters” on page 173
- “Master Broker” on page 176

Cluster Configuration Properties

You define a cluster by specifying *cluster configuration properties* for each of its member brokers. You can set these properties individually for each broker in the cluster, but it is generally more convenient to collect them into a central *cluster configuration file* that all of the brokers reference. This prevents the settings from getting out of agreement and ensures that all brokers in a cluster share the same, consistent configuration information.

The cluster configuration properties are described in detail in [Table 14–9](#). They include the following:

- `imq.cluster.brokerlist` gives the host names and port numbers for all brokers belonging to the cluster.
- `imq.cluster.masterbroker` designates which broker (if any) is the master broker that keeps track of state changes.
- `imq.cluster.url` specifies the location of the cluster configuration file, if any.

- `imq.cluster.hostname` gives the host name or IP address for the cluster connection service, used for internal communication between brokers in the cluster. This setting can be useful if more than one host is available: for example, if there is more than one network interface card in a computer.
- `imq.cluster.port` gives the port number for the cluster connection service.
- `imq.cluster.transport` specifies the transport protocol used by the cluster connection service, such as `tcp` or `ssl`.

The `hostname` and `port` properties can be set independently for each individual broker, but `brokerlist`, `masterbroker`, `url`, and `transport` must have the same values for all brokers in the cluster.

The following sections describe how to set a broker's cluster configuration properties, either individually for each broker in a cluster or centrally, using a cluster configuration file.

Setting Cluster Properties for Individual Brokers

You can set a broker's cluster configuration properties in its instance configuration file (or on the command line when you start the broker). For example, to create a cluster consisting of brokers at port 9876 on `host1`, port 5000 on `host2`, and the default port (7676) on `ctrlhost`, you would include the following property in the instance configuration files for all three brokers:

```
imq.cluster.brokerlist=host1:9876,host2:5000,ctrlhost
```

Notice that if you need to change the cluster configuration, this method requires you to update the instance configuration file for every broker in the cluster.

Using a Cluster Configuration File

For consistency and ease of maintenance, it is recommended that you collect all of the shared cluster configuration properties into a single cluster configuration file instead of setting them separately for each individual broker. In this method, each broker's instance configuration file must set the `imq.cluster.url` property to point to the location of the cluster configuration file: for example,

```
imq.cluster.url=file:/home/cluster.properties
```

The cluster configuration file then defines the shared configuration properties for all of the brokers in the cluster, such as the list of brokers to be connected (`imq.cluster.brokerlist`), the transport protocol to use for the cluster connection service (`imq.cluster.transport`), and optionally, the address of the master broker (`imq.cluster.masterbroker`). The following code defines the same cluster as in the previous example, with the broker running on `ctrlhost` serving as the master broker:

```
imq.cluster.brokerlist=host1:9876,host2:5000,ctrlhost  
imq.cluster.masterbroker=ctrlhost
```

Managing Clusters

This section describes how to connect a set of brokers to form a cluster, add new brokers to an existing cluster, and remove brokers from a cluster.

Connecting Brokers

There are two general methods of connecting brokers into a cluster: from the command line (using the `-cluster` option) or by setting the `imq.cluster.brokerlist` property in the cluster configuration file. Whichever method you use, each broker that you start attempts to connect to the other brokers every five seconds; the connection will succeed once the master broker is started up (if one is configured). If a broker in the cluster starts before the master broker, it will remain in a suspended state, rejecting client connections, until the master broker starts; the suspended broker then will automatically become fully functional.

To configure a broker cluster from the command line, use the `-cluster` option to the `imqbrokerd` command to specify the complete list of brokers in the cluster when you start each one. For example, the following command starts a new broker and connects it to the brokers running at the default port (7676) on `host1`, at port 5000 on `host2`, and at 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.

Whichever method you use, you must make sure that no broker in the cluster is given an address that resolves to the network loopback IP address (`127.0.0.1`). Any broker configured with this address will be unable to connect to other brokers in the cluster.

Note – Some Linux installers automatically set the `localhost` entry to the network loopback address. On such systems, you must modify the system 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 DNS lookup is performed before consulting the local hosts file. The line in `/etc/nsswitch.conf` should read as follows:

```
hosts: dns files
```

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 [“Message Encryption” on page 141](#). 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 any 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. Note that it is not necessary to issue this command to every broker in the cluster; executing it for any one broker will cause all of them to reload the cluster configuration.

- 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 [“Adding Brokers to a Cluster” on page 174](#), 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 177](#) 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.

Monitoring a Broker

This chapter describes the tools you can use to monitor a broker and how you can get metrics data. The chapter has the following sections:

- “Introduction to Monitoring Tools” on page 179
- “Configuring and Using Broker Logging” on page 181
- “Displaying Metrics Interactively” on page 186
- “Writing an Application to Monitor Brokers” on page 190

Reference information on specific metrics is available in [Chapter 18](#)

Introduction to Monitoring Tools

There are three monitoring interfaces for Message Queue™ information: log files, interactive commands, and a client API that can obtain metrics. Each has its advantages and disadvantages, as follows:

- Log files provide a long-term record of metrics data, but cannot easily be parsed.
- Commands enable you to quickly sample information tailored to your needs, but do not enable you to look at historical information or manipulate the data programmatically.
- The client API lets you extract information, process it, manipulate the data, present graphs or send alerts. However, to use it, you must write a custom application to capture and analyze the data.

[Table 10–1](#) compares the different tools.

TABLE 10-1 Benefits and Limitations of Metrics Monitoring Tools

Metrics Monitoring Tool	Benefits	Limitations
imqcmd metrics	Remote monitoring Convenient for spot checking Reporting interval set in command option; can be changed on the fly Easy to select specific data of interest Data presented in easy tabular format	No single command gets all data Difficult to analyze data programmatically Doesn't create historical record Difficult to see historical trends
Log files	Regular sampling Creates a historical record	Need to configure broker properties; must shut down and restart broker to take effect Local monitoring only Data format very difficult to read or parse; no parsing tools Reporting interval cannot be changed on the fly; the same for all metrics data Does not provide flexibility in selection of data Broker metrics only; destination and connection service metrics not included Possible performance hit if interval set too short
Client API	Remote monitoring Easy to select specific data of interest Data can be analyzed programmatically and presented in any format	Need to configure broker properties; must shut down and restart broker to take effect You need to write your own metrics monitoring client Reporting interval cannot be changed on the fly; the same for all metrics data

In addition to the differences shown in the table, each tool gathers a somewhat different subset of the metrics information generated by the broker. For information on which metrics data is gathered by each monitoring tool, see [Chapter 18](#).

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](#)):

```
.../instances/instanceName/log
```

Note – For a broker whose life cycle is controlled by the Application Server, the log files are located in a subdirectory of the domain directory for the domain for which the broker was started:

```
.../appServer_domainName_dir/imq/instances/imqbroker/log
```

The log files are simple text files. They are named as follows, from earliest to latest:

```
log.txt
log_1.txt
log_2.txt
...log_9.txt
```

By default, log files are rolled over once a week; the system maintains nine backup files.

- To change the directory in which the log files are kept, set the property `imq.log.file.dirpath` to the desired path.
- To change the root name of the log files from `log` to something else, set the `imq.log.file.filename` property.

The broker supports three log levels: `ERROR`, `WARNING`, `INFO`. [Table 10–2](#) explains each level.

TABLE 10-2 Logging Levels

Level	Description
ERROR	Messages indicating problems that could cause system failure.
WARNING	Alerts that should be heeded but will not cause system failure.
INFO	Reporting of metrics and other informational messages.

Setting a logging level gathers messages for that level and all higher levels. The default log level is INFO, so ERROR, WARNING, and INFO messages are all logged by default.

Log Message Format

A logged message consists of a time stamp, message code, and the message itself. The volume of information varies with the log level you have set. The following is an example of an INFO message.

```
[13/Sep/2000:16:13:36 PDT] [B1004]: Starting the broker service using tcp
[25374,100] with min threads 50 and max threads of 500
```

To change the time stamp time zone, see information about the `imq.log.timezone` property, which is described in [Table 14-8](#).

Changing the Logger Configuration

Log-related properties are described in [Table 14-8](#).

▼ 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. The following `imqbrokerd` options affect logging:

- metrics *interval* Logging interval for broker metrics, in seconds
- loglevel *level* Logging level (ERROR, WARNING, INFO, or NONE)
- silent Silent mode (no logging to console)
- tty Log all messages to console

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 181](#).

▼ 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](#)

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.

The following is an example of the log format for dead messages:

```
[29/Mar/2006:15:35:39 PST] [B1147]: Message 8-129.145.180.87(e7:6b:dd:5d:98:aa)-
35251-1143675279400 from destination Q:q0 has been placed on the DMQ because
[B0053]: Message on destination Q:q0 Expired: expiration time 1143675279402,
arrival time 1143675279401, JMSTimestamp 1143675279400
```

Dead message logging is disabled by default. To enable it, set the broker attribute `imq.destination.logDeadMsgs`.

Displaying Metrics Interactively

A Message Queue broker can report the following types of metrics:

- **Java Virtual Machine (JVM) metrics.** Information about the JVM heap size.
- **Brokerwide metrics.** Information about messages stored in a broker, message flows into and out of a broker, and memory use. Messages are tracked in terms of numbers of messages and numbers of bytes.
- **Connection Service metrics.** Information about connections and connection thread resources, and 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 189 for more information.

imqcmd metrics

The syntax and options of `imqcmd metrics` are shown in [Table 10–3](#) and [Table 10–4](#), respectively.

TABLE 10–3 `imqcmd metrics` Subcommand Syntax

Subcommand Syntax	Metrics Data Provided
<code>metrics bkr</code> [-b <i>hostName:portNumber</i>] [-m <i>metricType</i>] [-int <i>interval</i>] [-msp <i>numSamples</i>]	Displays broker metrics for the default broker or a broker at the specified host and port.

TABLE 10-3 `imqcmd metrics` Subcommand Syntax (Continued)

Subcommand Syntax	Metrics Data Provided
<pre>metrics svc -n serviceName [-b hostName:portNumber] [-m metricType] [-int interval] [-msp numSamples]</pre>	Displays metrics for the specified service on the default broker or on a broker at the specified host and port.
<pre>metrics dst -t destType -n destName [-b hostName:portNumber] [-m metricType] [-int interval] [-msp numSamples]</pre>	Displays metrics information for the physical destination of the specified type and name.

TABLE 10-4 `imqcmd metrics` Subcommand Options

Subcommand Options	Description
<code>-b hostName:portNumber</code>	Specifies the hostname and port of the broker for which metrics data is reported. The default is <code>localhost:7676</code> .
<code>-int interval</code>	Specifies the interval (in seconds) at which to display the metrics. The default is 5 seconds.
<code>-m metricType</code>	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).
<code>-msp numSamples</code>	Specifies the number of samples displayed in the output. The default is an unlimited number (infinite).
<code>-n destName</code>	Specifies the name of the physical destination (if any) for which metrics data is reported. There is no default.
<code>-n serviceName</code>	Specifies the connection service (if any) for which metrics data is reported. There is no default.
<code>-t destType</code>	Specifies the type (queue or topic) of the physical destination (if any) for which metrics data is reported. There is no default.

Using the metrics Subcommand to Display Metrics Data

This section describes the procedure for using the `metrics` subcommand to report metrics information.

▼ To Use the metrics Subcommand

- 1 Start the broker for which metrics information is desired.
See “Starting Brokers” on page 68.
- 2 Issue the appropriate `imqcmd metrics` subcommand and options as shown in Table 10–3 and Table 10–4.

Metrics Outputs: imqcmd metrics

This section contains examples of output for the `imqcmd metrics` subcommand. The examples show brokerwide, connection service, and physical destination metrics.

Brokerwide Metrics

To get the rate of message and packet flow into and out of the broker at 10 second intervals, use the `metrics bkr` subcommand:

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

This command produces output similar to the following (see data descriptions in Table 18–2):

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](#)):

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](#)):

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](#)):

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 10–5](#) along with a description of the metrics data provided by the command.

TABLE 10-5 imqcmd query Subcommand Syntax

Subcommand Syntax	Metrics Data Provided
query bkr [-b <i>hostName: portNumber</i>]	Information on the current number of messages and message bytes stored in broker memory and persistent store (see “Displaying Broker Information” on page 98).
or	
query svc -n <i>serviceName</i> [-b <i>hostName:portNumber</i>]	Information on the current number of allocated threads and number of connections for a specified connection service (see “Displaying Connection Service Information” on page 104).
or	
query dst -t <i>destType</i> -n <i>destName</i> [-b <i>hostName:portNumber</i>]	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 117).

Note – Because of the limited metrics data provided by `imqcmd query`, this tool is not represented in the tables presented in [Chapter 18](#).

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 10-6](#), along with the type of metrics messages delivered to each destination.

TABLE 10-6 Metrics Topic Destinations

Topic Name	Type of Metrics Messages
mq.metrics.broker	Broker metrics
mq.metrics.jvm	Java Virtual Machine metrics

TABLE 10–6 Metrics Topic Destinations (Continued)

Topic Name	Type of Metrics Messages
<code>mq.metrics.destination_list</code>	List of destinations and their types
<code>mq.metrics.destination.queue.monitoredDestinationName</code>	Destination metrics for queue of specified name
<code>mq.metrics.destination.topic.monitoredDestinationName</code>	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”](#) on page 192 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 [“User Authorization: The Access Control Properties File”](#) on page 135.

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 `user2`.

```
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 [“Message Encryption”](#) on page 141.

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 [Chapter 18](#).

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” on page 195](#)
- [“Factors Affecting Performance” on page 198](#)
- [“Adjusting Configuration To Improve Performance” on page 208](#)

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 tradeoffs 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 service 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 interrelated. If message throughput is high, that means messages are less likely to be backlogged in the broker, 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, broker processing speed, and client processing speed, to name a few.

In any case, there are several different aspects of performance. Which of them are most important to you generally depends on the requirements of a particular application.

Benchmarks

Benchmarking is the process of creating a test suite for your messaging application and of measuring message throughput or other aspects of performance for this test suite.

For example, you could create a test suite by which some number of producing clients, using some number of connections, sessions, and message producers, send persistent or nonpersistent messages of a standard size to some number of queues or topics (all depending on

your messaging application design) at some specified rate. Similarly, the test suite includes some number of consuming clients, using some number of connections, sessions, and message consumers (of a particular type) that consume the messages in the test suite's physical destinations using a particular acknowledgment mode.

Using your standard test suite you can measure the time it takes between production and consumption of messages or the average message throughput rate, and you can monitor the system to observe connection thread usage, message storage data, message flow data, and other relevant metrics. You can then ramp up the rate of message production, or the number of message producers, or other variables, until performance is negatively affected. 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 effect on performance (see [“Application Design Factors Affecting Performance” on page 200](#)), 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 effect on performance.

Conversely, you can keep application-based factors constant and change your broker configuration in some controlled way (for example, change connection properties, thread pool properties, JVM memory limits, limit behaviors, file-based versus JDBC-based persistence, and so forth) and note how these changes affect performance.

This benchmarking of your application provides information that can be valuable when you want to increase the performance of a deployed application by tuning your message service. A benchmark allows the effect of a change or a set of changes to be more accurately predicted.

As a general rule, benchmarks should be run in a controlled test environment and for a long enough period of time for your message service to stabilize. (Performance is negatively affected 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 baseline use patterns you need to monitor your message service 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 broker 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 Affecting 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.

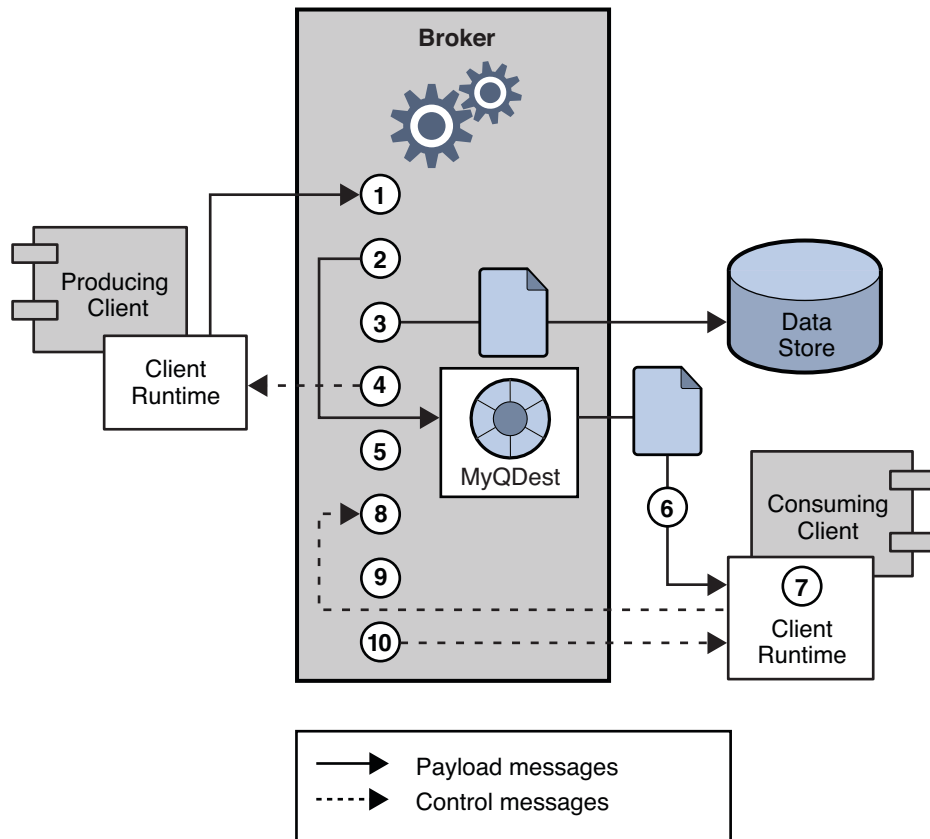


FIGURE 11-1 Message Delivery Through a Message Queue Service

▼ Message Delivery Steps

- 1 The message is delivered from producing client to broker.
- 2 The broker reads in the message.
- 3 The message is placed in persistent storage (for reliability).
- 4 The broker confirms receipt of the message (for reliability).
- 5 The broker determines the routing for the message.
- 6 The broker writes out the message.

- 7 The message is delivered from broker to consuming client.
- 8 The consuming client acknowledges receipt of the message (for reliability).
- 9 The broker processes client acknowledgment (for reliability).
- 10 The broker confirms that client acknowledgment has been processed.

Since these steps are sequential, any one of them can be a potential bottleneck in the delivery of messages from producing clients to consuming clients. Most of the steps depend on physical characteristics of the messaging system: network bandwidth, computer processing speeds, message service 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 effect 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 Affecting Performance

Application design decisions can have a significant effect on overall messaging performance.

The most important factors affecting performance are those that affect the reliability of message delivery. Among these are the following:

- [“Delivery Mode \(Persistent/Nonpersistent Messages\)” on page 201](#)
- [“Use of Transactions” on page 201](#)
- [“Acknowledgment Mode” on page 202](#)
- [“Durable and Nondurable Subscriptions” on page 203](#)

Other application design factors affecting performance are the following:

- [“Use of Selectors \(Message Filtering\)” on page 203](#)
- [“Message Size” on page 203](#)
- [“Message Body Type” on page 204](#)

The sections that follow describe the effect of each of these factors on messaging performance. As a general rule, there is a tradeoff 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—one high-reliability, low-performance, and one high-performance, low-reliability—and the choices of application design factors that characterize each. Between these extremes, there are many choices and tradeoffs that affect both reliability and performance.

TABLE 11-1 Comparison of High-Reliability and High-Performance Scenarios

Application DesignFactor	High-Reliability, Low-Performance Scenario	High-Performance, Low-Reliability Scenario
Delivery mode	Persistent messages	Nonpersistent messages
Use of transactions	Transacted sessions	No transactions
Acknowledgment mode	AUTO_ACKNOWLEDGE or CLIENT_ACKNOWLEDGE	DUPS_OK_ACKNOWLEDGE
Durable/nondurable subscriptions	Durable subscriptions	Nondurable 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

Delivery Mode (Persistent/Nonpersistent Messages)

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

Broker processing of persistent messages is slower than for nonpersistent messages for the following reasons:

- A broker must reliably store a persistent message so that it will not be lost should the broker fail.
- The broker must confirm receipt of each persistent message it receives. Delivery to the broker is guaranteed once the method producing the message returns without an exception.
- Depending on the client acknowledgment mode, the broker might need to confirm a consuming client's acknowledgment of a persistent message.

For both queues and topics with durable subscribers, performance was approximately 40% faster for nonpersistent messages. We obtained these results using 10k-sized messages and AUTO_ACKNOWLEDGE mode

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 nontransacted 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 broker.

If a session is closed without the client acknowledging the message or if the broker fails before the acknowledgment is processed, the broker redelivers that message, setting a `JMSRedelivered` flag.

For a nontransacted 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 broker 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 Nondurable Subscriptions

Subscribers to a topic destination fall into two categories, those with durable and nondurable subscriptions.

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

- The Message Queue message service must persistently store the list of messages assigned to each durable subscription so that should a broker fail, the list is available after recovery.
- Persistent messages for durable subscriptions are stored persistently, so that should a broker fail, the messages can still be delivered after recovery, when the corresponding consumer becomes active. By contrast, persistent messages for nondurable subscriptions are not stored persistently (should a broker fail, the corresponding consumer connection is lost and the message would never be delivered).

We compared performance for durable and nondurable subscribers in two cases: persistent and nonpersistent 10k-sized messages. Both cases use `AUTO_ACKNOWLEDGE` acknowledgment mode. We found an effect on performance only in the case of persistent messages which slowed durables by about 30%

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.

Creating 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.

In our tests, which compared throughput in kilobytes per second for 1k, 10k, and 100k-sized messages to a queue destination and `AUTO_ACKNOWLEDGE` acknowledgment mode, we found that nonpersistent messaging was about 50% faster for 1k messages, about 20% faster for 10k messages, and about 5% faster for 100k messages. The size of the message affected performance significantly for both persistent and nonpersistent messages. 100k messages are about 10 times faster than 10k, and 10k are about 5 times faster than 1k.

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 string.
- `StreamMessage` contains a stream of Java primitive values.
- `MapMessage` contains a set of name-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 Affecting 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 effect 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” on page 205](#)
- [“Operating System” on page 205](#)
- [“Java Virtual Machine \(JVM\)” on page 205](#)
- [“Connections” on page 205](#)
- [“Broker Limits and Behaviors” on page 207](#)
- [“Message Service Architecture” on page 206](#)
- [“Data Store Performance” on page 207](#)
- [“Client Runtime Configuration” on page 208](#)

The sections below describe the effect of each of these factors on messaging performance.

Hardware

For both the Message Queue broker 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 effect on the number of concurrent connections a broker 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 broker 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 broker 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 service can handle as well as the speed of message delivery.

Broker Connection Limits

All access to the broker 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 broker.

The number of connections to a broker is generally limited by the number of threads available. Message Queue can be configured to support either a dedicated thread model or a shared thread model (see [“Thread Pool Management” on page 77](#)).

The dedicated thread model is very fast because each connection has dedicated threads, however the number of connections is limited by the number of threads available (one input thread and one output thread for each connection). The shared thread model places no limit on the number of connections, however there is significant overhead and throughput delays in sharing threads among a number of connections, especially when those connections are busy.

Transport Protocols

Message Queue software allows clients to communicate with the broker using various low-level transport protocols. Message Queue supports the connection services (and corresponding protocols) described in [“Connection Services” on page 76](#).

The choice of protocols is based on application requirements (encrypted, accessible through a firewall), but the choice affects overall performance.

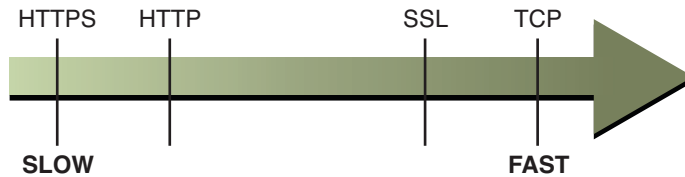


FIGURE 11-2 Transport Protocol Speeds

Our tests compared throughput for TCP and SSL for two cases: a high-reliability scenario (1k persistent messages sent to topic destinations with durable subscriptions and using `AUTO_ACKNOWLEDGE` acknowledgment mode) and a high-performance scenario (1k nonpersistent messages sent to topic destinations without durable subscriptions and using `DUPS_OK_ACKNOWLEDGE` acknowledgment mode).

In general we found that protocol has less effect 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. Additionally:

- TCP provides the fastest method to communicate with the broker.
- SSL is 50 to 70 percent slower than TCP when it comes to sending and receiving messages (50 percent for persistent messages, closer to 70 percent for nonpersistent messages). Additionally, establishing the initial connection is slower with SSL (it might take several seconds) because the client and broker (or Web Server in the case of HTTPS) need to establish a private key to be used when encrypting the data for transmission. The performance drop is caused by the additional processing required to encrypt and decrypt each low-level TCP packet.
- 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 Service Architecture

A Message Queue message service can be implemented as a single broker or as a cluster consisting of multiple interconnected broker instances.

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 service, 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](#)

Broker Limits and Behaviors

The message throughput that a broker might be required to handle is a function of the use patterns of the messaging applications the broker supports. However, the broker is limited in resources: memory, CPU cycles, and so forth. As a result, it would be possible for a broker to become overwhelmed to the point where it becomes unresponsive or unstable.

The Message Queue message broker has mechanisms built in for managing memory resources and preventing the broker from running out of memory. These mechanisms include configurable limits on the number of messages or message bytes that can be held by a broker or its individual physical destinations, and a set of behaviors that can be instituted when physical destination limits are reached.

With careful monitoring and tuning, these configurable mechanisms can be used to balance the inflow and outflow of messages so that system overload cannot occur. While these mechanisms consume overhead and can limit message throughput, they nevertheless maintain operational integrity.

Data Store Performance

Message Queue supports both file-based and JDBC-based persistence modules. File-based persistence uses individual files to store persistent data. JDBC-based persistence uses a Java Database Connectivity (JDBC™) interface and requires a JDBC-compliant data store. File-based persistence is generally faster than JDBC-based; however, some users prefer the redundancy and administrative control provided by a JDBC-compliant store.

In the case of file-based persistence, you can maximize reliability by specifying that persistence operations synchronize the in-memory state with the data store. This helps eliminate data loss due to system crashes, but at the expense of performance.

Client Runtime Configuration

The Message Queue client runtime provides client applications with an interface to the Message Queue message service. It supports all the operations needed for clients to send messages to physical destinations and to receive messages from such destinations. The client runtime is configurable (by setting connection factory attribute values), allowing you to control aspects of its behavior, such as connection flow metering, consumer flow limits, and connection flow limits, that can improve performance and message throughput. See [“Client Runtime Message Flow Adjustments” on page 213](#) for more information on these features and the attributes used to configure them.

Adjusting Configuration To Improve Performance

The following sections explain how configuration adjustments can affect 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 `imqsvcadm install` command. See [“Service Administrator Utility” on page 260 in Chapter 13](#)

In any case, verify settings by checking the broker's log file or using the `imqcmd metrics bkr -m cxn` command.

Tuning Transport Protocols

Once a protocol that meets application needs has been chosen, additional tuning (based on the selected protocol) might improve performance.

A protocol's performance can be modified using the following three broker properties:

- `imq.protocol.protocolType.nodelay`
- `imq.protocol.protocolType.inbufsz`
- `imq.protocol.protocolType.outbufsz`

For TCP and SSL protocols, these properties affect the speed of message delivery between client and broker. For HTTP and HTTPS protocols, these properties affect the speed of message delivery between the Message Queue tunnel servlet (running on a Web server) and the broker. For HTTP/HTTPS protocols there are additional properties that can affect performance (see [“HTTP/HTTPS Tuning” on page 210](#)).

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 timeout has occurred, the packet is sent.

For most messaging applications, performance is best if there is no delay in the sending of packets (Nagle's algorithm is not enabled). This is because most interactions between client and broker are request/response interactions: the client sends a packet of data to the broker and waits for a response. For example, typical interactions include:

- Creating a connection
- Creating a producer or consumer
- Sending a persistent message (the broker confirms receipt of the message)
- Sending a client acknowledgment in an `AUTO_ACKNOWLEDGE` or `CLIENT_ACKNOWLEDGE` session (the broker confirms processing of the acknowledgment)

For these interactions, most packets are smaller than the buffer size. This means that if Nagle's algorithm is used, the broker delays several milliseconds before sending a response to the consumer.

However, Nagle's algorithm may improve performance in situations where connections are slow and broker responses are not required. This would be the case where a client sends a nonpersistent message or where a client acknowledgment is not confirmed by the broker (`DUPS_OK_ACKNOWLEDGE` session).

inbufsz/outbufsz

The `inbufsz` property sets the size of the buffer on the input stream reading data coming in from a socket. Similarly, `outbufsz` sets the buffer size of the output stream used by the broker to write data to the socket.

In general, both parameters should be set to values that are slightly larger than the average packet being received or sent. A good rule of thumb is to set these property values to the size of the average packet plus 1 kilobyte (rounded to the nearest kilobyte). For example, if the broker is receiving packets with a body size of 1 kilobyte, the overall size of the packet (message body plus header plus properties) is about 1200 bytes; an `inbufsz` of 2 kilobytes (2048 bytes) gives reasonable performance. Increasing `inbufsz` or `outbufsz` greater than that size may improve performance slightly, but increases the memory needed for each connection.

HTTP/HTTPS Tuning

In addition to the general properties discussed in the previous two sections, HTTP/HTTPS performance is limited by how fast a client can make HTTP requests to the Web server hosting the Message Queue tunnel servlet.

A Web server might need to be optimized to handle multiple requests on a single socket. With JDK version 1.4 and later, HTTP connections to a Web server are kept alive (the socket to the Web server remains open) to minimize resources used by the Web server when it processes multiple HTTP requests. If the performance of a client application using JDK version 1.4 is slower than the same application running with an earlier JDK release, you might need to tune the Web server keep-alive configuration parameters to improve performance.

In addition to such Web server tuning, you can also adjust how often a client polls the Web server. HTTP is a request-based protocol. This means that clients using an HTTP-based protocol periodically need to check the Web server to see if messages are waiting. The `imq.httpjms.http.pullPeriod` broker property (and the corresponding `imq.httpsjms.https.pullPeriod` property) specifies how often the Message Queue client runtime polls the Web server.

If the `pullPeriod` value is `-1` (the default value), the client runtime polls the server as soon as the previous request returns, maximizing the performance of the individual client. As a result, each client connection monopolizes a request thread in the Web server, possibly straining Web server resources.

If the `pullPeriod` value is a positive number, the client runtime periodically sends requests to the Web server to see if there is pending data. In this case, the client does not monopolize a request thread in the Web server. Hence, if large numbers of clients are using the Web server, you might conserve Web server resources by setting the `pullPeriod` to a positive value.

Tuning the File-based Persistent Store

For information on tuning the file-based persistent store, see [“Persistence Services” on page 79](#).

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 basis or on a systemwide level (for all destinations, collectively).

Using Physical Destination Limits

For information on physical destination limits, see [Chapter 6](#)

Using Systemwide Limits

If message producers tend to overrun message consumers, messages can accumulate in the broker. The broker contains a mechanism for throttling back producers and swapping messages out of active memory in low memory conditions, but it is wise to set a hard limit on the total number of messages (and message bytes) that the broker can hold.

Control these limits by setting the `imq.system.max_count` and the `imq.system.max_size` broker properties.

For example:

```
imq.system.max_count=5000
```

The defined value above means that the broker will only hold up to 5000 undelivered/unacknowledged messages. If additional messages are sent, they are rejected by the broker. If a message is persistent then the producer will get an exception when it tries to send the message. If the message is nonpersistent, the broker silently drops the message.

When an exception is returned in sending a message, the client should pause for a moment and retry the send again. (Note that the exception will never be due to the broker's failure to receive a message; the only exceptions raised are those detected by the client on the sending side.)

Multiple Consumer Queue Performance

The efficiency with which multiple queue consumers process messages in a queue destination depends on the following configurable queue destination attributes:

- The number of active consumers (`maxNumActiveConsumers`)
- The maximum number of messages that can be delivered to a consumer in a single batch (`consumerFlowLimit`)

To achieve optimal message throughput there must be a sufficient number of active consumers to keep up with the rate of message production for the queue, and the messages in the queue must be routed and then delivered to the active consumers in such a way as to maximize their rate of consumption. The general mechanism for balancing message delivery among multiple consumers is described in the *Sun Java System™ Message Queue Technical Overview*.

If messages are accumulating in the queue, it is possible that there is an insufficient number of active consumers to handle the message load. It is also possible that messages are being delivered to the consumers in batch sizes that cause messages to be backing up on the consumers. For example, if the batch size (`consumerFlowLimit`) is too large, one consumer might receive all the messages in a queue while other active consumers receive none. If consumers are very fast, this might not be a problem.

However, if consumers are relatively slow, you want messages to be distributed to them evenly, and therefore you want the batch size to be small. The smaller the batch size, the more overhead

is required to deliver messages to consumers. Nevertheless, for slow consumers, there is generally a net performance gain to using small batch sizes.

Client Runtime Message Flow Adjustments

This section discusses flow control behaviors that affect performance (see [“Client Runtime Configuration” on page 208](#)). These behaviors are configured as attributes of connection factory administered objects. For information on setting connection factory attributes, see [Chapter 8](#)

Message Flow Metering

Messages sent and received by clients (*payload messages*), as well as Message Queue control messages, pass over the same client-broker connection. Delays in the delivery of control messages, such as broker acknowledgments, can result if control messages are held up by the delivery of payload messages. To prevent this type of congestion, Message Queue meters the flow of payload messages across a connection.

Payload messages are batched (as specified with the connection factory attribute `imqConnectionFactoryCount`) so that only a set number are delivered. After the batch has been delivered, delivery of payload messages is suspended and only pending control messages are delivered. This cycle repeats, as additional batches of payload messages are delivered followed by pending control messages.

The value of `imqConnectionFactoryCount` should be kept low if the client is doing operations that require many responses from the broker: for example, if the client is using `CLIENT_ACKNOWLEDGE` or `AUTO_ACKNOWLEDGE` mode, persistent messages, transactions, or queue browsers, or is adding or removing consumers. If, on the other hand, the client has only simple consumers on a connection using `DUPS_OK_ACKNOWLEDGE` mode, you can increase `imqConnectionFactoryCount` without compromising performance.

Message Flow Limits

There is a limit to the number of payload messages that the Message Queue client runtime can handle before encountering local resource limitations, such as memory. When this limit is approached, performance suffers. Hence, Message Queue lets you limit the number of messages per consumer (or messages per connection) that can be delivered over a connection and buffered in the client runtime, waiting to be consumed.

Consumer Flow Limits

When the number of payload messages delivered to the client runtime exceeds the value of `imqConsumerFlowLimit` for any consumer, message delivery for that consumer stops. It is resumed only when the number of unconsumed messages for that consumer drops below the value set with `imqConsumerFlowThreshold`.

The following example illustrates the use of these limits: consider the default settings for topic consumers:

```
imqConsumerFlowLimit=1000
imqConsumerFlowThreshold=50
```

When the consumer is created, the broker delivers an initial batch of 1000 messages (providing they exist) to this consumer without pausing. After sending 1000 messages, the broker stops delivery until the client runtime asks for more messages. The client runtime holds these messages until the application processes them. The client runtime then allows the application to consume at least 50% (`imqConsumerFlowThreshold`) of the message buffer capacity (i.e. 500 messages) before asking the broker to send the next batch.

In the same situation, if the threshold were 10%, the client runtime would wait for the application to consume at least 900 messages before asking for the next batch.

The next batch size is calculated as follows:

```
imqConsumerFlowLimit - ( current number of pending msgs in buffer
 )
```

So if `imqConsumerFlowThreshold` is 50%, the next batch size can fluctuate between 500 and 1000, depending on how fast the application can process the messages.

If the `imqConsumerFlowThreshold` is set too high (close to 100%), the broker will tend to send smaller batches, which can lower message throughput. If the value is set too low (close to 0%), the client may be able to finish processing the remaining buffered messages before the broker delivers the next set, again degrading message throughput. Generally speaking, unless you have specific performance or reliability concerns, you will not need to change the default value of `imqConsumerFlowThreshold` attribute.

The consumer-based flow controls (in particular, `imqConsumerFlowLimit`) are the best way to manage memory in the client runtime. Generally, depending on the client application, you know the number of consumers you need to support on any connection, the size of the messages, and the total amount of memory that is available to the client runtime.

Connection Flow Limits

In the case of some client applications, however, the number of consumers may be indeterminate, depending on choices made by end users. In those cases, you can still manage memory using connection-level flow limits.

Connection-level flow controls limit the total number of messages buffered for *all* consumers on a connection. If this number exceeds the value of `imqConnectionFlowLimit`, delivery of messages through the connection stops until that total drops below the connection limit. (The `imqConnectionFlowLimit` attribute is enabled only if you set `imqConnectionFlowLimitEnabled` to `true`.)

The number of messages queued up in a session is a function of the number of message consumers using the session and the message load for each consumer. If a client is exhibiting delays in producing or consuming messages, you can normally improve performance by redesigning the application to distribute message producers and consumers among a larger number of sessions or to distribute sessions among a larger number of connections.

Troubleshooting Problems

This chapter explains how to understand and resolve the following problems:

- [“A Client Cannot Establish a Connection” on page 217](#)
- [“Connection Throughput Is Too Slow” on page 221](#)
- [“A Client Cannot Create a Message Producer” on page 222](#)
- [“Message Production Is Delayed or Slowed” on page 223](#)
- [“Messages Are Backlogged” on page 226](#)
- [“Broker Throughput Is Sporadic” on page 230](#)
- [“Messages Are Not Reaching Consumers” on page 231](#)
- [“Dead Message Queue Contains Messages” on page 234](#)

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

Symptoms:

- Client cannot make a new connection.
- Client cannot auto-reconnect on failed connection.

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.

Possible cause: 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.

Possible cause: 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.

Possible cause: 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 [“Message Encryption” on page 141](#).

- If the status of a connection service is shown as paused, resume the service (see [“Pausing and Resuming a Connection Service” on page 106](#)).

Possible cause: Too few threads available for the number of connections required.

To confirm this cause of the problem: Check for the following entry in the broker log:

```
WARNING [B3004]: No threads are available to process a new connection on service
...
Closing the new connection.
```

Also check the number of connections on the connection service and the number of threads currently in use, using one of the following formats:

```
imqcmd query svc -n serviceName
imqcmd metrics svc -n serviceName -m cxn
```

Each connection requires two threads: one for incoming messages and one for outgoing messages (see [“Thread Pool Management” on page 77](#)).

To resolve the problem:

- If you are using a dedicated thread pool model (`imq.serviceName.threadpool_model=dedicated`), the maximum number of connections is half the maximum number of threads in the thread pool. Therefore, to increase the number of connections, increase the size of the thread pool (`imq.serviceName.max_threads`) or switch to the shared thread pool model.
- If you are using a shared thread pool model (`imq.serviceName.threadpool_model=shared`), the maximum number of connections is half the product of the connection monitor limit (`imq.serviceName.connectionMonitor_limit`) and the maximum number of threads (`imq.serviceName.max_threads`). Therefore, to increase the number of connections, increase the size of the thread pool or increase the connection monitor limit.
- Ultimately, the number of supportable connections (or the throughput on connections) will reach input/output limits. In such cases, use a multiple-broker cluster to distribute connections among the broker instances within the cluster.

Possible cause: Too few file descriptors for the number of connections required on the Solaris or Linux platform.

For more information about this issue, see [“Setting the File Descriptor Limit” on page 68](#).

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.

Possible cause: 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 the Windows platform there is a hard-coded backlog limit of 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 others. 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:

- 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.

Possible cause: 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 multiple-broker cluster.

Possible cause: Authentication or authorization of the user is failing.

The authentication may be failing for any of the following reasons:

- Incorrect password
- No entry for user in user repository
- User does not have access permission for 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 for the user 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 whether there is an entry for the user.
- Check the access control properties file to see whether there are restrictions on access to the connection service.

To resolve the problem:

- If the wrong password was used, provide the correct password.
- If there is no entry for the user in the user repository, add one (see [“Populating and Managing a User Repository” on page 131](#)).
- If the user does not have access permission for the connection service, edit the access control properties file to grant such permission (see [“Access Control for Connection Services” on page 138](#)).

Connection Throughput Is Too Slow

Symptoms:

- 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 217](#), but rather by message input/output rates.

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 that 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.](#)

Possible cause: [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 consult a network administrator.
- 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: Upgrade the network link.

Possible cause: Connection service protocol is inherently slow compared to TCP.

For example, SSL-based or HTTP-based protocols are slower than TCP (see [“Transport Protocols” on page 206](#)).

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 you can do other than attempt to tune the protocol as described in [“Tuning Transport Protocols” on page 209](#).

Possible cause: Connection service protocol is not optimally tuned.

To confirm this cause of the problem: Try tuning the protocol to see whether it makes a difference.

To resolve the problem: Try tuning the protocol, as described in [“Tuning Transport Protocols” on page 209](#).

Possible cause: Messages are so large that 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.

Possible cause: 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 what appears to be slow connection throughput cannot be explained by any of the causes above, see [“Factors Affecting Performance” on page 198](#) for other possible bottlenecks and check for symptoms associated with the following problems:

- [“Message Production Is Delayed or Slowed” on page 223](#)
- [“Messages Are Backlogged” on page 226](#)
- [“Broker Throughput Is Sporadic” on page 230](#)

To resolve the problem: Follow the problem resolution guidelines provided in the troubleshooting sections listed above.

A Client Cannot Create a Message Producer

Symptom:

- A message producer cannot be created for a physical destination; the client receives an exception.

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.

Possible cause: 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:

```
imqcmd query dst
```

(see [“Displaying Information about Physical Destinations” on page 117](#)). 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 the exception

```
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 118](#)).

Possible cause: 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 exception

```
JMSSecurityException [C4076]: Client does not have permission to create producer
on destination
```

and makes the following entries in the broker log:

```
[B2041]: Producer on destination denied
```

```
[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 139](#)).

Message Production Is Delayed or Slowed

Symptoms:

- When sending persistent messages, the `send` method does not return and the client blocks.
- When sending a persistent message, the client receives an exception.
- A producing client slows down.

Possible causes:

- The broker 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.

Possible cause: The broker is backlogged and has responded by slowing message producers.

A backlogged broker accumulates messages in broker memory. When the number of messages or 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 message bytes in brokerwide memory (for all physical destinations) reaches configured limits, the broker will attempt to conserve memory resources by rejecting the newest messages. Also, when system memory limits are reached because physical destination or brokerwide limits have not been set properly, the broker takes increasingly serious action to prevent memory overload. These actions include throttling back message producers.

To confirm this cause of the problem: When a message is rejected by the broker because of configured message limits, the broker returns the exception

`JMSEException [C4036]: A server error occurred`

and makes the following entry in the broker log:

`[B2011]: Storing of JMS message from IMQconn failed`

This message is followed by another indicating the limit that has been reached:

`[B4120]: Cannot store message on destination destName because capacity of maxNumMsgs would be exceeded.`

if the exceeded message limit is on a physical destination, or

`[B4024]: The maximum number of messages currently in the system has been exceeded, rejecting message.`

if the limit is brokerwide.

More generally, you can check for message limit conditions before the rejections occur as follows:

- Query physical destinations and the broker and inspect their configured message limit settings.
- Monitor the number of messages or message bytes currently in a physical destination or in the broker as a whole, using the appropriate `imqcmd` commands. See [Chapter 18](#) for information about metrics you can monitor and the commands you use to obtain them.

To resolve the problem:

- Modify the message limits on a physical destination (or brokerwide), being careful not to exceed memory resources.

In general, you should manage memory at the individual destination level, so that brokerwide message limits are never reached. For more information, see [“Broker Adjustments” on page 211](#).

- Change the limit behaviors on a destination so as not to 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](#)).

Possible cause: 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 it, the producing client is blocked. This condition can also occur if destination or brokerwide message limits are reached, as described above.

To confirm this cause of the problem: If the broker is unable to write to the data store, it makes one of the following entries in the broker log:

```
[B2011]: Storing of JMS message from connectionID failed
[B4004]: Failed to persist message messageID
```

To resolve the problem:

- In the case of file-based persistence, try increasing the disk space of the file-based data store.
- In the case of a JDBC-compliant data store, check that JDBC-based persistence is properly configured (see [“Configuring a Persistent Data Store” on page 91](#)). If so, consult your database administrator to troubleshoot other database problems.

Possible cause: Broker acknowledgment timeout is too short.

Because of slow connections or a lethargic broker (caused by high CPU utilization or scarce memory resources), a broker may 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 exception

```
JMSEException [C4000]: Packet acknowledge failed
```

To resolve the problem: Change the value of the `imqAckTimeout` connection factory attribute (see [“Reliability And Flow Control” on page 161](#)).

Possible cause: 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 208).

Messages Are Backlogged

Symptoms:

- Message production is delayed or produced messages are rejected by the broker.
- Messages take an unusually long time to reach consumers.
- The number of messages or message bytes in the broker (or in specific destinations) increases steadily over time.

To see whether messages are accumulating, check how the number of messages or message bytes in the broker changes over time and compare to configured limits. First check the configured limits:

```
imqcmd query bkr
```

Note – The `imqcmd metrics bkr` subcommand does not display this information.

Then check for message accumulation in each destination:

```
imqcmd list dst
```

To see whether messages have exceeded configured destination or brokerwide limits, check the broker log for the entry

```
[B2011]: Storing of JMS message from ... failed.
```

This entry will be followed by another identifying the limit that has been exceeded.

Possible causes:

- [There are inactive durable subscriptions on a topic destination.](#)
- [Too few consumers are 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.](#)

Possible cause: [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:

- Purge all messages for the offending durable subscriptions (see [“Managing Durable Subscriptions” on page 108](#)).
- Specify message limit and limit behavior attributes for the topic (see [Table 15–1](#)). 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 120](#)).
- Limit the time messages can remain in memory by rewriting 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 `imqOverrideJMSEExpiration` and `imqJMSEExpiration` connection factory attributes (see [“Message Header Overrides” on page 300](#)).

Possible cause: Too few consumers are 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: 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 212](#)).
- Specify message limit and limit behavior attributes for the queue (see [Table 15–1](#)). 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 120](#)).
- Limit the time messages can remain in memory by rewriting 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 `imqOverrideJMSEExpiration` and `imqJMSEExpiration` connection factory attributes (see [“Message Header Overrides” on page 300](#)).

Possible cause: 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 are getting backlogged with messages because of 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 212](#)).

Possible cause: 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 may 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 may be a problem.
- Check to see whether the client has received the following exception:

```
JMSEException [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 213](#)).

Possible cause: 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
- Persistent store
- JVM memory limits

To confirm this cause of the problem: Check that none of the other possible causes of this problem are responsible.

To resolve the problem:

- Upgrade the speed of your computer or data store.
- Use a broker cluster to distribute the load among multiple broker instances.

Possible cause: Client code defects; consumers are not acknowledging messages.

Messages are held in a destination until they have been acknowledged by all consumers to which they 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 the `CLIENT_ACKNOWLEDGE` acknowledgment mode or transacted session may not be calling `Session.acknowledge` or `Session.commit` regularly.
- Consumers using the `AUTO_ACKNOWLEDGE` acknowledgment mode may 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. Messages under this header were sent to consumers but not acknowledged. If this number is the same as the total number of messages, then 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.

Broker Throughput Is Sporadic

Symptom:

- Message throughput sporadically drops and then resumes normal performance.

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.](#)

Possible cause: 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; this can cause the broker to become 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 208](#)).
- Increase system swap space.

Possible cause: 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 longer 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 when memory reclamation takes place.

To resolve the problem: In computers with multiple CPUs, set the memory reclamation to take place in parallel:

```
-XX:+UseParallelGC=true
```

Possible cause: 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 possible causes of this problem are responsible.

To resolve the problem: Let the system run for awhile; performance should improve.

Messages Are Not Reaching Consumers

Symptom:

- Messages sent by producers are not received by consumers.

Possible causes:

- [Limit behaviors are causing messages to be deleted on the broker.](#)
- [Message timeout value is expiring.](#)
- [Clocks are not synchronized.](#)
- [Consuming client failed to start message delivery on a connection.](#)

Possible cause: Limit behaviors are causing messages to be deleted on the broker.

When the number of messages or message bytes in destination memory reach configured limits, the broker attempts to conserve memory resources. Three of the configurable behaviors adopted by the broker when these limits are reached will cause messages to be lost:

- REMOVE_OLDEST: Delete the oldest messages.
- REMOVE_LOW_PRIORITY: Delete the lowest-priority messages according to age.
- REJECT_NEWEST: Reject new persistent messages.

To confirm this cause of the problem: Check the dead message queue, as described under [“Dead Message Queue Contains Messages”](#) on page 234. Specifically, use the instructions under [“The number of messages, or their sizes, exceed destination limits.”](#) 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
```

Possible cause: Message timeout value is expiring.

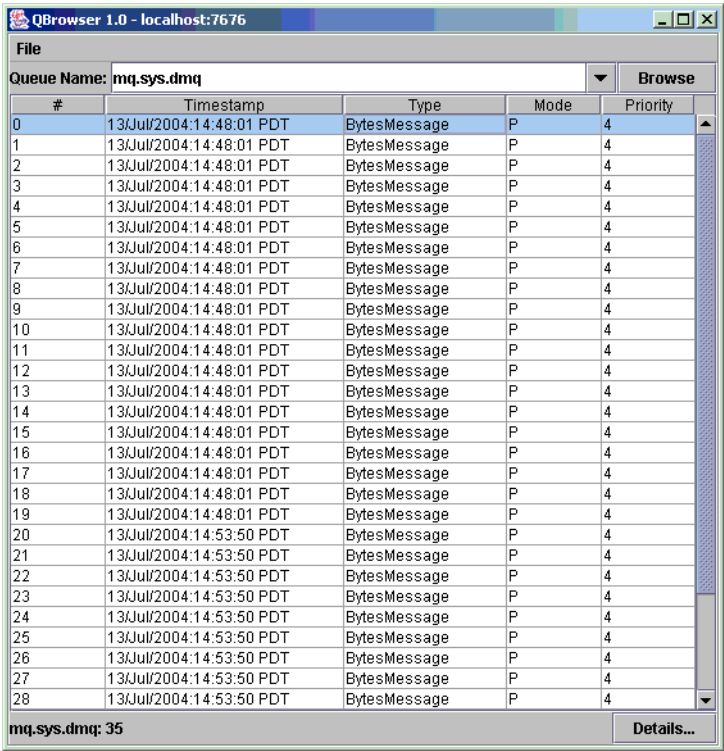
The broker deletes messages whose timeout value has expired. If a destination gets sufficiently backlogged with messages, messages whose time-to-live value is too short may be deleted.

To confirm this cause of the problem: Use the QBrowser demo application to look at the contents of the dead message queue and see whether messages are timing out. For the QBrowser demo’s platform-specific location, see [Appendix A](#) and look in the tables for “Example Applications and Locations.”

Here is an example invocation on the Windows platform:

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

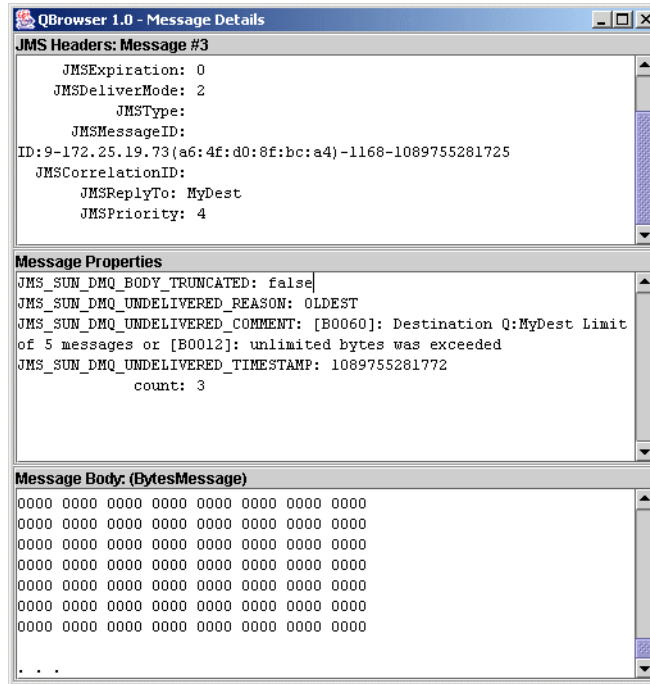
When the QBrowser main window appears, select the queue name `mq.sys.dmq` and then click Browse. A list like the following appears:



The screenshot shows the QBrowser 1.0 application window. The title bar reads "QBrowser 1.0 - localhost:7676". The "File" menu is open, and the "Queue Name" field is set to "mq.sys.dmq". A "Browse" button is visible. Below the menu, a table lists 29 messages. The table has columns for #, Timestamp, Type, Mode, and Priority. All messages are of type "BytesMessage" with mode "P" and priority "4". The timestamps range from 13/Jul/2004:14:48:01 PDT to 13/Jul/2004:14:53:50 PDT. At the bottom of the window, the status bar shows "mq.sys.dmq: 35" and a "Details..." button.

#	Timestamp	Type	Mode	Priority
0	13/Jul/2004:14:48:01 PDT	BytesMessage	P	4
1	13/Jul/2004:14:48:01 PDT	BytesMessage	P	4
2	13/Jul/2004:14:48:01 PDT	BytesMessage	P	4
3	13/Jul/2004:14:48:01 PDT	BytesMessage	P	4
4	13/Jul/2004:14:48:01 PDT	BytesMessage	P	4
5	13/Jul/2004:14:48:01 PDT	BytesMessage	P	4
6	13/Jul/2004:14:48:01 PDT	BytesMessage	P	4
7	13/Jul/2004:14:48:01 PDT	BytesMessage	P	4
8	13/Jul/2004:14:48:01 PDT	BytesMessage	P	4
9	13/Jul/2004:14:48:01 PDT	BytesMessage	P	4
10	13/Jul/2004:14:48:01 PDT	BytesMessage	P	4
11	13/Jul/2004:14:48:01 PDT	BytesMessage	P	4
12	13/Jul/2004:14:48:01 PDT	BytesMessage	P	4
13	13/Jul/2004:14:48:01 PDT	BytesMessage	P	4
14	13/Jul/2004:14:48:01 PDT	BytesMessage	P	4
15	13/Jul/2004:14:48:01 PDT	BytesMessage	P	4
16	13/Jul/2004:14:48:01 PDT	BytesMessage	P	4
17	13/Jul/2004:14:48:01 PDT	BytesMessage	P	4
18	13/Jul/2004:14:48:01 PDT	BytesMessage	P	4
19	13/Jul/2004:14:48:01 PDT	BytesMessage	P	4
20	13/Jul/2004:14:53:50 PDT	BytesMessage	P	4
21	13/Jul/2004:14:53:50 PDT	BytesMessage	P	4
22	13/Jul/2004:14:53:50 PDT	BytesMessage	P	4
23	13/Jul/2004:14:53:50 PDT	BytesMessage	P	4
24	13/Jul/2004:14:53:50 PDT	BytesMessage	P	4
25	13/Jul/2004:14:53:50 PDT	BytesMessage	P	4
26	13/Jul/2004:14:53:50 PDT	BytesMessage	P	4
27	13/Jul/2004:14:53:50 PDT	BytesMessage	P	4
28	13/Jul/2004:14:53:50 PDT	BytesMessage	P	4

Double-click any message to display details about that message:



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.

Possible cause: 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 the problem: Check that you are running a time synchronization program, as described in [“Preparing System Resources”](#) on page 67.

Possible cause: 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.

Dead Message Queue Contains Messages

Symptom:

- 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 Total	Msgs Count	UnAck	Avg Size

MyDest	Queue	RUNNING	0	0	5	0	1177.0
mq.sys.dm	Queue	RUNNING	0	0	35	0	1422.0

Successfully listed destinations.

In this example, the dead message queue, mq.sys.dm, contains 35 messages.

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 messages before they 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.
- Consumers are failing to acknowledge messages.
- Durable consumers are inactive.
- An unexpected broker error has occurred.

Possible cause: 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. For the QBrowser demo’s platform-specific location, see [Appendix A](#) and look in the tables for “Example Applications and Locations.”

Here is an example invocation on the Windows platform:

```
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 earlier under “[Message timeout value is expiring](#)” appears. Double-click any message to display details about that message, as shown under “[Message timeout value is expiring](#).”

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 the problem: Increase the destination limits. For example:

```
imqcmd update dst -n MyDest -o maxNumMsgs=1000
```

Possible cause: 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 the problem: Check that you are running a time synchronization program, as described in “[Preparing System Resources](#)” on page 67.

Possible cause: Consumers are not receiving messages before they 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.

Possible cause: 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 may be overwhelming consumption rate.

To resolve the problem: Add more consumer clients or set the destination's limit behavior to `FLOW_CONTROL` (which uses consumption rate to control production rate), using a command such as the following:

```
imqcmd update dst -n myDst -t q -o consumerFlowLimit=FLOW_CONTROL
```

Possible cause: 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's limit behavior to `FLOW_CONTROL` (which uses consumption rate to control production rate), using a command such as the following:

```
imqcmd update dst -n myDst -t q -o consumerFlowLimit=FLOW_CONTROL
```

Use metrics to examine the destination's input and output, using a command such as the following:

```
imqcmd metrics dst -n myDst -t q -m rts
```

In the metrics output, examine the following values:

- **Msgs/sec Out:** 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:** 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 so, there is a discrepancy between the processing speeds of producers and consumers. You can also check the number of unacknowledged (`UnAked`) messages sent, 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 has occurred.](#)

Possible cause: A consumer is too slow.

To confirm this cause of the problem: Use metrics to determine the rate of production and consumption, as described above under "[Producers are faster than consumers.](#)"

To resolve the problem:

- Set the destinations' limit behavior to `FLOW_CONTROL`, using a command such as 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, 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, increase destination limits as described in the next item.
- Increase destination limits based on number of messages or bytes, or both. To change the number of messages on a destination, enter a command with the following format:

```
imqcmd update dst -n destName -t {q|t} -o maxNumMsgs=number
```

To change the size of a destination, enter a command with 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 periods of high production load.

Possible cause: 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 so, list the active transactions as follows:

```
imqcmd list txn
```

Here 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 the problem: Contact application developers to fix the coding error.

Possible cause: 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 it uses client-based acknowledgment (CLIENT_ACKNOWLEDGE), first decrease the number of messages stored on the client, using 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 Total	Msgs Count	UnAck	Avg Size
MyDest	Queue	RUNNING	0	0	5	200	1177.0
mq.sys.dm	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 this 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.

Possible cause: 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.

Possible cause: An unexpected broker error has occurred.

To confirm this cause of the problem: Use QBrowser to examine a message, as described earlier under “[Producers are faster than consumers.](#)” 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](#)
- [Chapter 14](#)
- [Chapter 15](#)
- [Chapter 16](#)
- [Chapter 17](#)
- [Chapter 18](#)

Command Line Reference

This chapter provides reference information on the use of the Message Queue™ command line administration utilities. It consists of the following sections:

- “Command Line Syntax” on page 243
- “Broker Utility” on page 244
- “Command Utility” on page 248
- “Object Manager Utility” on page 256
- “Database Manager Utility” on page 257
- “User Manager Utility” on page 259
- “Service Administrator Utility” on page 260
- “Key Tool Utility” on page 261

Command Line Syntax

Message Queue command line utilities are shell commands. The name of the utility is a command and its subcommands or options are arguments passed to that command. There is no need for separate commands to start or quit the utility.

All the command line utilities share the following command syntax:

```
utilityName [subcommand] [commandArgument] [ [-optionName [optionArgument]] ... ]
```

where *utilityName* is one of the following:

- `imqbrokerd` (Broker utility)
- `imqcmd` (Command utility)
- `imqobjmgr` (Object Manager utility)
- `imqdbmgr` (Database Manager utility)
- `imqusermgr` (User Manager utility)
- `imqsvcadm` (Service Administrator utility)
- `imqkeytool` (Key Tool utility)

Subcommands and command-level arguments, if any, must precede all options and their arguments; the options themselves may appear in any order. All subcommands, command arguments, options, and option arguments are separated with spaces. If the value of an option argument contains a space, the entire value must be enclosed in quotation marks. (It is generally safest to enclose any attribute-value pair in quotation marks.)

The following command, which starts the default broker, is an example of a command line with no subcommand clause:

```
imqbrokerd
```

Here is a fuller example:

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

This command destroys a queue destination (destination type q) named myQueue. Authentication is performed on the user name admin; the command will prompt for a password. The command will be performed without prompting for confirmation (-f option) and in silent mode, without displaying any output (-s option).

Broker Utility

The Broker utility (imqbrokerd) starts a broker. Command line options override values in the broker configuration files, but only for the current broker session.

Table 13–1 shows the options to the imqbrokerd command and the configuration properties, if any, overridden by each option.

TABLE 13–1 Broker Utility Options

Option	Properties Overridden	Description
-name <i>instanceName</i>	imq.instanceName	Instance name of broker Multiple broker instances running on the same host must have different instance names. Default value: imqbroker
-port <i>portNumber</i>	imq.portmapper.port	Port number for broker's Port Mapper Message Queue clients use this port number to connect to the broker. Multiple broker instances running on the same host must have different Port Mapper port numbers. Default value: 7676

TABLE 13-1 Broker Utility Options (Continued)

Option	Properties Overridden	Description
<code>-cluster broker1 [[,broker2] ...]</code>	<code>imq.cluster.brokerlist</code>	<p>Connect brokers into cluster¹</p> <p>The specified brokers are merged with the list in the <code>imq.cluster.brokerlist</code> property. Each broker argument has one of the forms</p> <p style="margin-left: 40px;"><i>hostName:portNumber</i></p> <p style="margin-left: 40px;"><i>hostName</i></p> <p style="margin-left: 40px;"><i>:portNumber</i></p> <p>If <i>hostName</i> is omitted, the default value is <code>localhost</code>; if <i>portNumber</i> is omitted, the default value is 7676.</p>
<code>-Dproperty=value</code>	Corresponding property in instance configuration file	<p>Set configuration property</p> <p>See Chapter 14 for information about broker configuration properties.</p> <p>Caution: Be careful to check the spelling and formatting of properties set with this option. Incorrect values will be ignored without notification or warning.</p>
<code>-reset props</code>	None	<p>Reset configuration properties</p> <p>Replaces the broker's existing instance configuration file <code>config.properties</code> with an empty file; all properties assume their default values.</p>
<code>-reset store</code>	None	<p>Reset persistent data store</p> <p>Clears all persistent data from the data store (including persistent messages, durable subscriptions, and transaction information), allowing you to start the broker instance with a clean slate. To prevent the persistent store from being reset on subsequent restarts, restart the broker instance without the <code>-reset</code> option.</p> <p>To clear only persistent messages or durable subscriptions, use <code>-reset messages</code> or <code>-reset durables</code> instead.</p>
<code>-reset messages</code>	None	Clear persistent messages from data store
<code>-reset durables</code>	None	Clear durable subscriptions from data store

¹ Applies only to broker clusters

TABLE 13–1 Broker Utility Options (Continued)

Option	Properties Overridden	Description
-backup <i>fileName</i>	None	Back up configuration change record to file ¹ See “ Managing the Configuration Change Record ” on page 176 for more information.
-restore <i>fileName</i>	None	Restore configuration change record from backup file ¹ The backup file must have been previously created using the -backup option. See “ Managing the Configuration Change Record ” on page 176 for more information.
-remove instance	None	Remove broker instance ² Deletes the instance configuration file, log files, persistent store, and other files and directories associated with the instance.
-password <i>keyPassword</i>	imq.keystore.password	Password for SSL certificate key store ³
-dbuser <i>userName</i>	imq.persist.jdbc.user	User name for JDBC-based persistent data store
-dbpassword <i>dbPassword</i>	imq.persist.jdbc.password	Password for JDBC-based persistent data store ³
-ldappassword <i>ldapPassword</i>	imq.user_repository.ldap.password	Password for LDAP user repository ³
-passfile <i>filePath</i>	imq.passfile.enabled imq.passfile.dirpath imq.passfile.name	Location of password file Sets the broker’s imq.passfile.enabled property to true, imq.passfile.dirpath to the path containing the password file, and imq.passfile.name to the file name itself. See “ Password Files ” on page 149 for more information.
-shared	imq.jms.threadpool_model	Use shared thread pool model to implement jms connection service Execution threads will be shared among connections to increase the number of connections supported. Sets the broker’s imq.jms.threadpool_model property to shared.

¹ Applies only to broker clusters² Requires user confirmation unless -force is also specified³ This option is deprecated and will eventually be removed. Either omit the password (so that the user will be prompted for it interactively) or use the -passfile option to specify a file containing the password.

TABLE 13–1 Broker Utility Options (Continued)

Option	Properties Overridden	Description
-javahome <i>path</i>	None	Location of alternative Java runtime Default behavior: Use runtime installed on system or bundled with Message Queue.
-vmargs <i>arg1</i> [[<i>arg2</i>] ...]	None	Pass arguments to Java virtual machine Arguments are separated with spaces. To pass more than one argument, or an argument containing a space, enclose the argument list in quotation marks. VM arguments can be passed only from the command line; there is no associated configuration property in the instance configuration file.
-license [<i>licenseName</i>]	None	License to load, if different from default for installed edition of Message Queue product: pe: Platform Edition with basic features try: Platform Edition with enterprise features (90-day trial) unl: Enterprise Edition If no license name is specified, all licenses installed on the system are listed.
-upgrade-store-nobackup	None	Automatically remove old data store on upgrade to Message Queue 3.5 or 3.5 SPx from an incompatible version ² See the <i>Message Queue Installation Guide</i> for more information.
-force	None	Perform action without user confirmation This option applies only to the -remove instance and -upgrade-store-nobackup options, which normally require confirmation.
-loglevel <i>level</i>	imq.broker.log.level	Logging level: NONE ERROR WARNING INFO Default value: INFO
-metrics <i>interval</i>	imq.metrics.interval	Logging interval for broker metrics, in seconds

² Requires user confirmation unless -force is also specified

TABLE 13–1 Broker Utility Options (Continued)

Option	Properties Overridden	Description
-tty	imq.log.console.output	Log all messages to console Sets the broker's imq.log.console.output property to ALL. If not specified, only error and warning messages will be logged.
-s -silent	imq.log.console.output	Silent mode (no logging to console) Sets the broker's imq.log.console.output property to NONE.
-version	None	Display version information ⁴
-h -help	None	Display usage help ⁴

⁴ Any other options specified on the command line are ignored.

Command Utility

The Command utility (imqcmd) is used for managing brokers, connection services, connections, physical destinations, durable subscriptions, and transactions.

All imqcmd commands must include a subcommand (except those using the -v or -h option to display product version information or usage help). The possible subcommands are listed here and described in detail in the corresponding sections below. In all cases, if the subcommand accepts a broker address (-b option) and no host name or port number is specified, the values localhost and 7676 are assumed by default.

“Broker Management” on page 250	
shutdown bkr	Shut down broker
restart bkr	Restart broker
pause bkr	Pause broker
resume bkr	Resume broker
update bkr	Set broker properties
reload cls	Reload cluster configuration
query bkr	List broker property values
metrics bkr	Display broker metrics

[“Connection Service Management” on page](#)

pause svc	Pause connection service
resume svc	Resume connection service
update svc	Set connection service properties
list svc	List connection services available on broker
query svc	List connection service property values
metrics svc	Display connection service metrics

[“Connection Management” on page 252](#)

list cxn	List connections on broker
query cxn	Display connection information

[“Physical Destination Management” on page 252](#)

create dst	Create physical destination
destroy dst	Destroy physical destination
pause dst	Pause message delivery for physical destination
resume dst	Resume message delivery for physical destination
update dst	Set physical destination properties
purge dst	Purge all messages from physical destination
compact dst	Compact physical destination
list dst	List physical destinations
query dst	List physical destination property values
metrics dst	Display physical destination metrics

[“Durable Subscription Management” on page 254](#)

destroy dur	Destroy durable subscription
purge dur	Purge all messages for durable subscription
list dur	List durable subscriptions for topic

[“Transaction Management” on page 255](#)

commit txn	Commit transaction
rollback txn	Roll back transaction
list txn	List transactions being tracked by broker

query txn	Display transaction information
-----------	---------------------------------

Broker Management

The Command utility cannot be used to start a broker; use the Broker utility (`imqbrokerd`) instead. Once the broker is started, you can use the `imqcmd` subcommands listed in [Table 13–2](#) to manage and control it.

TABLE 13–2 Command Utility Subcommands for Broker Management

Syntax	Description
shutdown bkr [-b <i>hostName:portNumber</i>]	Shut down broker
restart bkr [-b <i>hostName:portNumber</i>]	Restart broker Shuts down the broker and then restarts it using the same options specified when it was originally started.
pause bkr [-b <i>hostName:portNumber</i>]	Pause broker See “Pausing a Broker” on page 100 for more information.
resume bkr [-b <i>hostName:portNumber</i>]	Resume broker
update bkr [-b <i>hostName:portNumber</i>] -o <i>property1=value1</i> [[-o <i>property2=value2</i>] ...]	Set broker properties See Chapter 14 for information on broker properties.
reload cls	Reload cluster configuration ¹ Forces all persistent information to be brought up to date.
query bkr -b <i>hostName:portNumber</i>	List broker property values Also lists all running brokers connected to the specified broker in a cluster.

¹ Applies only to broker clusters

TABLE 13–2 Command Utility Subcommands for Broker Management (Continued)

Syntax	Description
<pre>metrics bkr [-b <i>hostName:portNumber</i>] [-m <i>metricType</i>] [-int <i>interval</i>] [-msp <i>numSamples</i>]</pre>	<p>Display broker metrics</p> <p>The <code>-m</code> option specifies the type of metrics to display:</p> <ul style="list-style-type: none"> <code>t t l</code>: Messages and packets flowing into and out of the broker <code>r t s</code>: Rate of flow of messages and packets into and out of the broker per second <code>c x n</code>: Connections, virtual memory heap, and threads <p>Default value: <code>t t l</code>.</p> <p>The <code>-int</code> option specifies the interval, in seconds, at which to display metrics. Default value: 5.</p> <p>The <code>-msp</code> option specifies the number of samples to display. Default value: Unlimited (infinite).</p>

Connection Service Management

Table 13–3 lists the `imqcmd` subcommands for managing connection services.

TABLE 13–3 Command Utility Subcommands for Connection Service Management

Syntax	Description
<pre>pause svc -n <i>serviceName</i> [-b <i>hostName:portNumber</i>]</pre>	<p>Pause connection service</p> <p>The <code>admin</code> connection service cannot be paused.</p>
<pre>resume svc -n <i>serviceName</i> [-b <i>hostName:portNumber</i>]</pre>	<p>Resume connection service</p>
<pre>update svc -n <i>serviceName</i> [-b <i>hostName:portNumber</i>] -o <i>property1=value1</i> [[-o <i>property2=value2</i>] ...]</pre>	<p>Set connection service properties</p> <p>See “Connection Properties” on page 263 for information on connection service properties.</p>
<pre>list svc [-b <i>hostName:portNumber</i>]</pre>	<p>List connection services available on broker</p>
<pre>query svc -n <i>serviceName</i> [-b <i>hostName:portNumber</i>]</pre>	<p>List connection service property values</p>

TABLE 13–3 Command Utility Subcommands for Connection Service Management (Continued)

Syntax	Description
<code>metrics svc -n serviceName</code> <code>[-b hostName:portNumber]</code> <code>[-m metricType]</code> <code>[-int interval]</code> <code>[-msp numSamples]</code>	<p>Display connection service metrics</p> <p>The <code>-m</code> option specifies the type of metrics to display:</p> <ul style="list-style-type: none"><code>ttl</code>: Messages and packets flowing into and out of the broker by way of the specified connection service<code>rts</code>: Rate of flow of messages and packets into and out of the broker per second by way of the specified connection service<code>cxn</code>: Connections, virtual memory heap, and threads <p>Default value: <code>ttl</code>.</p> <p>The <code>-int</code> option specifies the interval, in seconds, at which to display metrics. Default value: 5.</p> <p>The <code>-msp</code> option specifies the number of samples to display. Default value: Unlimited (infinite).</p>

Connection Management

Table 13–4 lists the `imqcmd` subcommands for managing connections.

TABLE 13–4 Command Utility Subcommands for Connection Service Management

Syntax	Description
<code>list cxn [-svn serviceName]</code> <code>[-b hostName:portNumber]</code>	<p>List connections on broker</p> <p>Lists all connections on the broker to the specified connection service. If no connection service is specified, all connections are listed.</p>
<code>query cxn -n connectionID</code> <code>[-b hostName:portNumber]</code>	<p>Display connection information</p>

Physical Destination Management

Table 13–5 lists the `imqcmd` subcommands for managing physical destinations. In all cases, the `-t` (destination type) option can take either of two values:

- q: Queue destination
- t: Topic destination

TABLE 13–5 Command Utility Subcommands for Physical Destination Management

Syntax	Description
<pre>create dst -t <i>destType</i> -n <i>destName</i> [-o <i>property1=value1</i>] [[-o <i>property2=value2</i>] ...]</pre>	<p>Create physical destination¹</p> <p>The destination name <i>destName</i> may contain only alphanumeric characters (no spaces) and must begin with an alphabetic character or the underscore (_) or dollar sign (\$) character. It may not begin with the characters mq.</p>
<pre>destroy dst -t <i>destType</i> -n <i>destName</i></pre>	<p>Destroy physical destination¹</p> <p>This operation cannot be applied to a system-created destination, such as a dead message queue.</p>
<pre>pause dst [-t <i>destType</i> -n <i>destName</i>] [-pst <i>pauseType</i>]</pre>	<p>Pause message delivery for physical destination</p> <p>Pauses message delivery for the physical destination specified by the -t and -n options. If these options are not specified, all destinations are paused.</p> <p>The -pst option specifies the type of message delivery to be paused:</p> <ul style="list-style-type: none"> CONSUMERS: Pause delivery to message consumers PRODUCERS: Pause delivery to message producers ALL: Pause all message delivery <p>Default value: ALL</p>
<pre>resume dst [-t <i>destType</i> -n <i>destName</i>]</pre>	<p>Resume message delivery for physical destination</p> <p>Resumes message delivery for the physical destination specified by the -t and -n options. If these options are not specified, all destinations are resumed.</p>
<pre>update dst -t <i>destType</i> -n <i>destName</i> -o <i>property1=value1</i> [[-o <i>property2=value2</i>] ...]</pre>	<p>Set physical destination properties</p> <p>See Chapter 15 for information on physical destination properties.</p>
<pre>purge dst -t <i>destType</i> -n <i>destName</i></pre>	<p>Purge all messages from physical destination</p>
<pre>compact dst [-t <i>destType</i> -n <i>destName</i>]</pre>	<p>Compact physical destination</p> <p>Compacts the file-based persistent data store for the physical destination specified by the -t and -n options. If these options are not specified, all destinations are compacted.</p> <p>A destination must be paused before it can be compacted.</p>

¹ Cannot be performed in a broker cluster whose master broker is temporarily unavailable

TABLE 13–5 Command Utility Subcommands for Physical Destination Management (Continued)

Syntax	Description
<code>list dst [-t <i>destType</i>] [-tmp]</code>	List physical destinations Lists all physical destinations of the type specified by the <code>-t</code> option. If no destination type is specified, both queue and topic destinations are listed. If the <code>-tmp</code> option is specified, temporary destinations are listed as well.
<code>query dst -t <i>destType</i> -n <i>destName</i></code>	List physical destination property values
<code>metrics dst -t <i>destType</i> -n <i>destName</i> [-m <i>metricType</i>] [-int <i>interval</i>] [-msp <i>numSamples</i>]</code>	Display physical destination metrics The <code>-m</code> option specifies the type of metrics to display: <i>ttl</i> : Messages and packets flowing into and out of the destination and residing in memory <i>rts</i> : Rate of flow of messages and packets into and out of the broker per second, along with other rate information <i>con</i> : Metrics related to message consumers <i>dsk</i> : Disk usage Default value: <i>ttl</i> . The <code>-int</code> option specifies the interval, in seconds, at which to display metrics. Default value: 5. The <code>-msp</code> option specifies the number of samples to display. Default value: Unlimited (infinite).

Durable Subscription Management

Table 13–6 lists the `imqcmd` subcommands for managing durable subscriptions.

TABLE 13–6 Command Utility Subcommands for Durable Subscription Management

Syntax	Description
<code>destroy dur -c <i>clientID</i> -n <i>subscriberName</i></code>	Destroy durable subscription ¹
<code>purge dur -c <i>clientID</i> -n <i>subscriberName</i></code>	Purge all messages for durable subscription
<code>list dur -d <i>topicName</i></code>	List durable subscriptions for topic

¹ Cannot be performed in a broker cluster whose master broker is temporarily unavailable

Transaction Management

Table 13–7 lists the `imqcmd` subcommands for managing transactions.

TABLE 13–7 Command Utility Subcommands for Transaction Management

Syntax	Description
<code>commit txn -n transactionID</code>	Commit transaction
<code>rollback txn -n transactionID</code>	Roll back transaction
<code>list txn</code>	List transactions being tracked by broker
<code>query txn -n transactionID</code>	Display transaction information

General Command Utility Options

The additional options listed in Table 13–8 can be applied to any subcommand of the `imqcmd` command.

TABLE 13–8 General Command Utility Options

Option	Description
<code>-secure</code>	Use secure connection to broker with <code>ssladmin</code> connection service
<code>-u userName</code>	User name for authentication If this option is omitted, the Command utility will prompt for it interactively.
<code>-p password</code>	Password for authentication ¹
<code>-passfile path</code>	Location of password file See “Password Files” on page 149 for more information.
<code>-rtm timeoutInterval</code>	Initial timeout interval, in seconds This is the initial length of time that the Command utility will wait for a reply from the broker before retrying a request. Each subsequent retry will use a timeout interval that is a multiple of this initial interval. Default value: 10.
<code>-rtr numRetries</code>	Number of retries to attempt after a broker request times out Default value: 5.

¹ This option is deprecated and will eventually be removed. Either omit the password (so that the user will be prompted for it interactively) or use the `-passfile` option to specify a file containing the password.

TABLE 13–8 General Command Utility Options (Continued)

Option	Description
- javahome <i>path</i>	Location of alternative Java runtime Default behavior: Use runtime installed on system or bundled with Message Queue.
- f	Perform action without user confirmation
- s	Silent mode (no output displayed)
- v	Display version information ^{2,3}
- h	Display usage help ^{2,3}
- H	Display expanded usage help, including attribute list and examples ^{2,3}

² Any other options specified on the command line are ignored.

³ User name and password not needed

Object Manager Utility

The Object Manager utility (`imqobjmgr`) creates and manages Message Queue administered objects. [Table 13–9](#) lists the available subcommands.

TABLE 13–9 Object Manager Subcommands

Subcommand	Description
add	Add administered object to object store
delete	Delete administered object from object store
list	List administered objects in object store
query	Display administered object information
update	Modify administered object

[Table 13–10](#) lists the options to the `imqobjmgr` command.

TABLE 13–10 Object Manager Options

Option	Description
- l <i>lookupName</i>	JNDI lookup name of administered object
- j <i>attribute=value</i>	Attributes of JNDI object store (see “Object Stores” on page 153)

TABLE 13–10 Object Manager Options (Continued)

Option	Description
-t <i>objectType</i>	Type of administered object: q: Queue destination t: Topic destination cf: Connection factory qf: Queue connection factory tf: Topic connection factory xcf: Connection factory for distributed transactions xqf: Queue connection factory for distributed transactions xtf: Topic connection factory for distributed transactions e: SOAP endpoint (see <i>Message Queue Developer's Guide for Java Clients</i>)
-o <i>attribute=value</i>	Attributes of administered object (see “Administered Object Attributes” on page 156 and Chapter 16)
-r <i>readOnlyState</i>	Is administered object read-only? If true, client cannot modify object's attributes. Default value: false.
-i <i>fileName</i>	Name of command file containing all or part of subcommand clause
-pre	Preview results without performing command This option is useful for checking the values of default attributes.
-javahome <i>path</i>	Location of alternative Java runtime Default behavior: Use runtime installed on system or bundled with Message Queue.
-f	Perform action without user confirmation
-s	Silent mode (no output displayed)
-v	Display version information ¹
-h	Display usage help ¹
-H	Display expanded usage help, including attribute list and examples ¹

¹ Any other options specified on the command line are ignored.

Database Manager Utility

The Database Manager utility (`imqdbmgr`) sets up the database schema for a JDBC-based persistent data store. You can also use it to delete Message Queue database tables that have become corrupted or to change the data store. Table 13–11 lists the available subcommands.

TABLE 13–11 Database Manager Subcommands

Subcommand	Description
create all	Create new database and persistent store schema Used on embedded database systems. The broker property <code>imq.persist.jdbc.createdburl</code> must be specified.
create tbl	Create persistent store schema for existing database Used on external database systems.
delete tbl	Delete Message Queue database tables from current persistent store
delete oldtbl	Delete Message Queue database tables from earlier-version persistent store Used after the persistent store has been automatically migrated to the current version of Message Queue.
recreate tbl	Re-create persistent store schema Deletes all existing Message Queue database tables from the current persistent store and then re-creates the schema.
reset lck	Reset persistent store lock Resets the lock so that the persistent store database can be used by other processes.

Table 13–12 lists the options to the `imqdbmgr` command.

TABLE 13–12 Database Manager Options

Option	Description
-b <i>instanceName</i>	Instance name of broker
-D <i>property= value</i>	Set broker configuration property See “ Persistence Properties ” on page 270 for information about persistence-related broker configuration properties. Caution: Be careful to check the spelling and formatting of properties set with this option. Incorrect values will be ignored without notification or warning.
-u <i>name</i>	User name for authentication
-p <i>password</i>	Password for authentication ¹

¹ This option is deprecated and will eventually be removed. Either omit the password (so that the user will be prompted for it interactively) or use the `-passfile` option to specify a file containing the password.

TABLE 13–12 Database Manager Options (Continued)

Option	Description
-passfile <i>path</i>	Location of password file See “Password Files” on page 149 for more information.
-v	Display version information ²
-h	Display usage help ²

² Any other options specified on the command line are ignored.

User Manager Utility

The User Manager utility (`imqusermgr`) is used for populating or editing a flat-file user repository. The utility must be run on the same host where the broker is installed; if a broker-specific user repository does not yet exist, you must first start up the corresponding broker instance in order to create it. You will also need the appropriate permissions to write to the repository: on the Solaris or Linux platforms, this means you must be either the root user or the user who originally created the broker instance.

Table 13–13 lists the subcommands available with the `imqusermgr` command. In all cases, the `-i` option specifies the instance name of the broker to whose user repository the command applies; if not specified, the default name `imqbroker` is assumed.

TABLE 13–13 User Manager Subcommands

Syntax	Description
add [-i <i>instanceName</i>] -u <i>userName</i> -p <i>password</i> [-g <i>group</i>]	Add user and password to repository The optional -g option specifies a group to which to assign this user: admin user anonymous
delete [-i <i>instanceName</i>] -u <i>userName</i>	Delete user from repository
update [-i <i>instanceName</i>] -u <i>userName</i> -p <i>password</i> [-a <i>activeState</i>] update [-i <i>instanceName</i>] -u <i>userName</i> -a <i>activeState</i> [-p <i>password</i>]	Set user's password or active state (or both) The -a option takes a boolean value specifying whether to make the user active (<code>true</code>) or inactive (<code>false</code>). Default value: <code>true</code> .

TABLE 13–13 User Manager Subcommands <i>(Continued)</i>	
Syntax	Description
<code>list [-i instanceName] [-u userName]</code>	Display user information If no user name is specified, all users in the repository are listed.

In addition, the options listed in [Table 13–14](#) can be applied to any subcommand of the `imqusermgr` command.

TABLE 13–14 General User Manager Options	
Option	Description
<code>-f</code>	Perform action without user confirmation
<code>-s</code>	Silent mode (no output displayed)
<code>-v</code>	Display version information ¹
<code>-h</code>	Display usage help ¹

¹ Any other options specified on the command line are ignored.

Service Administrator Utility

The Service Administrator utility (`imqsvcadmin`) installs a broker as a Windows service. [Table 13–15](#) lists the available subcommands.

TABLE 13–15 Service Administrator Subcommands	
Subcommand	Description
<code>install</code>	Install service
<code>remove</code>	Remove service
<code>query</code>	Display startup options Startup options can include whether the service is started manually or automatically, its location, the location of the Java runtime, and the values of arguments passed to the broker on startup (see Table 13–16).

[Table 13–16](#) lists the options to the `imqsvcadmin` command.

TABLE 13–16 Service Administrator Options

Option	Description
-j <code>avahome path</code>	Location of alternative Java runtime Default behavior: Use runtime installed on system or bundled with Message Queue.
-j <code>rehome path</code>	Location of alternative Java Runtime Environment (JRE)
-vmargs <i>arg1</i> [<i>arg2</i> ...]	Additional arguments to pass to Java Virtual Machine (JVM) running broker service ¹ Example: <code>imqsvcadm install -vmargs "-Xms16m -Xmx128m"</code>
-args <i>arg1</i> [<i>arg2</i> ...]	Additional command line arguments to pass to broker service ¹ Example: <code>imqsvcadm install -args "-passfile d:\\imqpassfile"</code> See “ Broker Utility ” on page 244 for information about broker command line arguments.
-h	Display usage help ²

¹ These arguments can also be specified in the Start Parameters field under the General tab in the service’s Properties window (reached via the Services tool in the Windows Administrative Tools control panel).

² Any other options specified on the command line are ignored.

Any information you specify using the -j`avahome`, -vmargs, and -args options is stored in the Windows registry under the keys JREHome, JVMArgs, and ServiceArgs in the path

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\IMQ_Broker\Parameters

Key Tool Utility

The Key Tool utility (`imqkeytool`) generates a self-signed certificate for the broker, which can be used for the `ssljms`, `ssladmin`, or `cluster` connection service. The syntax is

```
imqkeytool -broker
```

On UNIX systems, you may need to run the utility from the superuser (root) account.

Broker Properties Reference

This chapter provides reference information about configuration properties for a message broker. It consists of the following sections:

- “Connection Properties” on page 263
- “Routing Properties” on page 265
- “Persistence Properties” on page 270
- “Security Properties” on page 274
- “Monitoring Properties” on page 278
- “Cluster Configuration Properties” on page 282
- “Alphabetical List of Broker Properties” on page 283

Connection Properties

Table 14–1 lists the broker properties related to connection services.

TABLE 14–1 Broker Connection Properties

Property	Type	Default	Description
<code>imq.service.activelist</code>	String	<code>jms, admin</code>	List of connection services to be activated at broker startup, separated by commas
<code>imq.hostname</code>	String	All available IP addresses	Default host name or IP address for all connection services
<code>imq.portmapper.hostname</code>	String	None	Host name or IP address of Port Mapper If specified, overrides <code>imq.hostname</code> .

TABLE 14–1 Broker Connection Properties (Continued)

Property	Type	Default	Description
<code>imq.portmapper.port</code> ¹	Integer	7676	Port number of Port Mapper Note – If multiple broker instances are running on the same host, each must be assigned a unique Port Mapper port.
<code>imq.serviceName.protocolType.hostname</code> ²	String	None	Host name or IP address for connection service If specified, overrides <code>imq.hostname</code> for the designated connection service.
<code>imq.serviceName.protocolType.port</code> ²	Integer	0	Port number for connection service A value of 0 specifies that the port number should be allocated dynamically by the Port Mapper.
<code>imq.portmapper.backlog</code>	Integer	50	Maximum number of pending Port Mapper requests in operating system backlog
<code>imq.serviceName.threadpool_model</code> ³	String	dedicated	Threading model for thread pool management: dedicated: Two dedicated threads per connection, one for incoming and one for outgoing messages shared: Connections processed by shared thread when sending or receiving messages The dedicated model limits the number of connections that can be supported, but provides higher performance; the shared model increases the number of possible connections, but at the cost of lower performance because of the additional overhead needed for thread management.
<code>imq.serviceName.min_threads</code>	Integer	jms: 10 ssljms: 10 httpjms: 10 httpsjms: 10 admin: 4 ssladmin: 4	Minimum number of threads maintained in connection service's thread pool When the number of available threads exceeds this threshold, threads will be shut down as they become free until the minimum is reached. The default value varies by connection service, as shown.

¹ Can be used with `imqcmd update bkr` command

² jms, ssljms, admin, and ssladmin services only; see [Appendix C](#) for information on configuring the httpjms and httpsjms services

³ jms and admin services only

TABLE 14–1 Broker Connection Properties (Continued)

Property	Type	Default	Description
<code>imq.serviceName.max_threads</code>	Integer	jms: 1000 ssljms: 500 httpjms: 500 httpsjms: 500 admin: 10 ssladmin: 10	Number of threads beyond which no new threads are added to the thread pool for use by the named connection service Must be greater than 0 and greater than the value of <code>imq.serviceName.min_threads</code> . The default value varies by connection service, as shown.
<code>imq.shared.connectionMonitor_limit</code> ⁴	Integer	Solaris: 512 Linux: 512 Windows: 64	Maximum number of connections monitored by a distributor thread The system allocates enough distributor threads to monitor all connections. The smaller the value of this property, the faster threads can be assigned to active connections. A value of -1 denotes an unlimited number of connections per thread. The default value varies by operating-system platform, as shown.
<code>imq.ping.interval</code>	Integer	120	Interval, in seconds, at which to test connection between client and broker A value of 0 or -1 disables periodic testing of the connection.

⁴ Shared threading model only

Routing Properties

Table 14–2 lists the broker properties related to routing services. Properties that configure the automatic creation of destinations are listed in Table 14–3.

TABLE 14–2 Broker Routing Properties

Property	Type	Default	Description
<code>imq.system.max_count</code> ¹	Integer	-1	Maximum number of messages held by broker A value of -1 denotes an unlimited message count.

¹ Can be used with `imqcmd update bkr` command

TABLE 14–2 Broker Routing Properties (Continued)

Property	Type	Default	Description
<code>imq.system.max_size¹</code>	String	- 1	<p>Maximum total size of messages held by broker</p> <p>The value may be expressed in bytes, kilobytes, or megabytes, using the following suffixes:</p> <ul style="list-style-type: none"> b: Bytes k: Kilobytes (1024 bytes) m: Megabytes (1024 × 1024 = 1,048,576 bytes) <p>An unsuffixed value is expressed in bytes; a value of - 1 denotes an unlimited message capacity.</p> <p>Examples:</p> <ul style="list-style-type: none"> 1600: 1600 bytes 1600b: 1600 bytes 16k: 16 kilobytes (= 16,384 bytes) 16m: 16 megabytes (= 16,777,216 bytes) - 1: No limit
<code>imq.message.max_size¹</code>	String	70m	<p>Maximum size of a single message body</p> <p>The syntax is the same as for <code>imq.system.max_size</code> (see above).</p>
<code>imq.message.expiration.interval</code>	Integer	60	Interval, in seconds, at which expired messages are reclaimed
<code>imq.resourceState.threshold</code>	Integer	green: 0 yellow: 80 orange: 90 red: 98	Percent utilization at which memory resource state is triggered (where <i>resourceState</i> is green, yellow, orange, or red)
<code>imq.resourceState.count</code>	Integer	green: 5000 yellow: 500 orange: 50 red: 0	<p>Maximum number of incoming messages allowed in a batch before checking whether memory resource state threshold has been reached (where <i>resourceState</i> is green, yellow, orange, or red)</p> <p>This limit throttles back message producers as system memory becomes increasingly scarce.</p>
<code>imq.destination.DMQ.truncateBody¹</code>	Boolean	false	<p>Remove message body before storing in dead message queue?</p> <p>If <code>true</code>, only the message header and property data will be saved.</p>
<code>imq.transaction.autorollback</code>	Boolean	false	<p>Automatically roll back distributed transactions left in prepared state at broker startup?</p> <p>If <code>false</code>, transactions must be manually committed or rolled back using the Command utility (<code>imqcmd</code>).</p>

¹ Can be used with `imqcmd update bkr` command

TABLE 14-3 Broker Properties for Auto-Created Destinations

Property	Type	Default	Description
<code>imq.autocreate.queue^{1,2}</code>	Boolean	<code>true</code>	Allow auto-creation of queue destinations?
<code>imq.autocreate.topic³</code>	Boolean	<code>true</code>	Allow auto-creation of topic destinations?
<code>imq.autocreate.destination.maxNumMsgs</code>	Integer	<code>100000</code>	Maximum number of unconsumed messages A value of -1 denotes an unlimited number of messages.
<code>imq.autocreate.destination.maxBytesPerMsg</code>	String	<code>10k</code>	Maximum size, in bytes, of any single message The value may be expressed in bytes, kilobytes, or megabytes, using the following suffixes: b: Bytes k: Kilobytes (1024 bytes) m: Megabytes (1024 × 1024 = 1,048,576 bytes) An unsuffixed value is expressed in bytes; a value of -1 denotes an unlimited message size. Examples: 1600: 1600 bytes 1600b: 1600 bytes 16k: 16 kilobytes (= 16,384 bytes) 16m: 16 megabytes (= 16,777,216 bytes) -1: No limit
<code>imq.autocreate.destination.maxTotalMsgBytes</code>	String	<code>10m</code>	Maximum total memory, in bytes, for unconsumed messages The syntax is the same as for <code>imq.autocreate.destination.maxBytesPerMsg</code> (see above).

¹ Can be used with `imqcmd update bkr` command² Queue destinations only³ Topic destinations only

TABLE 14-3 Broker Properties for Auto-Created Destinations (Continued)

Property	Type	Default	Description
<code>imq.autocreate.destination.limitBehavior</code>	String	REJECT_NEWEST	<p>Broker behavior when memory-limit threshold reached:</p> <p>FLOW_CONTROL: Slow down producers</p> <p>REMOVE_OLDEST: Throw out oldest messages</p> <p>REMOVE_LOW_PRIORITY: Throw out lowest-priority messages according to age; no notification to producing client</p> <p>REJECT_NEWEST: Reject newest messages; notify producing client with an exception only if message is persistent</p> <p>If the value is REMOVE_OLDEST or REMOVE_LOW_PRIORITY and the <code>imq.autocreate.destination.useDMQ</code> property is true, excess messages are moved to the dead message queue.</p>
<code>imq.autocreate.destination.maxNumProducers</code>	Integer	100	<p>Maximum number of message producers for destination</p> <p>When this limit is reached, no new producers can be created. A value of -1 denotes an unlimited number of producers.</p>
<code>imq.autocreate.queue.maxNumActiveConsumers</code> ²	Integer	1	<p>Maximum number of active message consumers in load-balanced delivery from queue destination</p> <p>A value of -1 denotes an unlimited number of consumers.</p>
<code>imq.autocreate.queue.maxNumBackupConsumers</code> ²	Integer	0	<p>Maximum number of backup message consumers in load-balanced delivery from queue destination</p> <p>A value of -1 denotes an unlimited number of consumers.</p>

² Queue destinations only

TABLE 14-3 Broker Properties for Auto-Created Destinations (Continued)

Property	Type	Default	Description
<code>imq.autocreate.queue.consumerFlowLimit²</code>	Integer	1000	<p>Maximum number of messages delivered to queue consumer in a single batch</p> <p>In load-balanced queue delivery, this is the initial number of queued messages routed to active consumers before load balancing begins. A destination consumer can override this limit by specifying a lower value on a connection.</p> <p>A value of -1 denotes an unlimited number of messages.</p>
<code>imq.autocreate.topic.consumerFlowLimit³</code>	Integer	1000	<p>Maximum number of messages delivered to topic consumer in a single batch</p> <p>A value of -1 denotes an unlimited number of consumers.</p>
<code>imq.autocreate.destination.isLocalOnly</code>	Boolean	false	<p>Local delivery only?</p> <p>This property applies only to destinations in broker clusters, and cannot be changed once the destination has been created. If <code>true</code>, 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).</p>
<code>imq.autocreate.queue.localDeliveryPreferred²</code>	Boolean	false	<p>Local delivery preferred?</p> <p>This property applies only to load-balanced queue delivery in broker clusters. If <code>true</code>, messages will be delivered to remote consumers only if there are no consumers on the local broker; the destination must not be restricted to local-only delivery (<code>imq.autocreate.destination.isLocalOnly</code> must be <code>false</code>).</p>
<code>imq.autocreate.destination.usedMQ</code>	Boolean	true	<p>Send dead messages to dead message queue?</p> <p>If <code>false</code>, dead messages will simply be discarded.</p>

² Queue destinations only³ Topic destinations only

Persistence Properties

Message Queue™ supports both file-based and JDBC-based models for persistent data storage. The broker property `imq.persist.store` (Table 14–4) specifies which model to use. The following sections describe the broker configuration properties for the two models.

TABLE 14–4 Global Broker Persistence Property

Property	Type	Default	Description
<code>imq.persist.store</code>	String	<code>file</code>	Model for persistent data storage: <code>file</code> : File-based persistence <code>jdbc</code> : JDBC-based persistence

File-Based Persistence

Table 14–5 lists the broker properties related to file-based persistence.

TABLE 14–5 Broker Properties for File-Based Persistence

Property	Type	Default	Description
<code>imq.persist.file.message.max_record_size</code>	String	<code>1m</code>	Maximum-size message to add to message storage file Any message exceeding this size will be stored in a separate file of its own. The value may be expressed in bytes, kilobytes, or megabytes, using the following suffixes: b : Bytes k : Kilobytes (1024 bytes) m : Megabytes (1024 × 1024 = 1,048,576 bytes) An unsuffixed value is expressed in bytes. Examples: 1600 : 1600 bytes 1600b : 1600 bytes 16k : 16 kilobytes (= 16,384 bytes) 16m : 16 megabytes (= 16,777,216 bytes)

TABLE 14–5 Broker Properties for File-Based Persistence (Continued)

Property	Type	Default	Description
<code>imq.persist.file.destination.message.filepool.limit</code>	Integer	100	<p>Maximum number of free files available for reuse in destination file pool</p> <p>Free files in excess of this limit will be deleted. The broker will create and delete additional files in excess of the limit as needed.</p> <p>The higher the limit, the faster the broker can process persistent data.</p>
<code>imq.persist.file.message.filepool.cleanratio</code>	Integer	0	<p>Percentage of files in free file pools to be maintained in a clean (empty) state</p> <p>The higher this value, the less disk space is required for the file pool, but the more overhead is needed to clean files during operation.</p>
<code>imq.persist.file.message.cleanup</code>	Boolean	false	<p>Clean up files in free file pools on shutdown?</p> <p>Setting this property to <code>true</code> saves disk space for the file store, but slows broker shutdown.</p>
<code>imq.persist.file.sync.enabled</code>	Boolean	false	<p>Synchronize in-memory state with physical storage device?</p> <p>Setting this property to <code>true</code> eliminates data loss due to system crashes, but at a cost in performance.</p> <p>Note – If running Sun Cluster and the Sun Cluster Data Service for Message Queue, set this property to <code>true</code> for brokers on all cluster nodes.</p>

JDBC-Based Persistence

Table 14–6 lists the broker properties related to JDBC-based persistence. Examples shown are for the PointBase® family of database products from DataMirror Mobile Solutions, Inc.

TABLE 14-6 Broker Properties for JDBC-Based Persistence

Property	Example	Description
<code>imq.persist.jdbc.brokerid</code>	Not required for PointBase embedded version	<p>(Optional) Broker instance identifier</p> <p>Must be an alphanumeric string of no more than $n - 12$ characters, where n is the maximum table name length allowed by the database.</p> <p>This identifier is appended to database table names to make them unique in the case where more than one broker instance is using the same database as a persistent data store. It is usually unnecessary for an embedded database, which stores data for only one broker instance.</p>
<code>imq.persist.jdbc.driver</code>	<code>com.pointbase.jdbc.jdbcUniversalDriver</code>	Java class name of JDBC driver for connecting to database
<code>imq.persist.jdbc.opendburl</code>	<code>jdbc:pointbase:embedded:dbName;</code> <code>database.home=</code> <code>.../instances/instanceName/dbstore</code>	URL for opening connection to existing database
<code>imq.persist.jdbc.createdburl</code>	<code>jdbc:pointbase:embedded:dbName;</code> <code>new,database.home=</code> <code>.../instances/instanceName/dbstore</code>	<p>(Optional) URL for creating new database</p> <p>Needed only if the database will be created using the Message Queue Database Manager utility (<code>imqdbmgr</code>).</p>
<code>imq.persist.jdbc.closedburl</code>	Not required for PointBase	(Optional) URL for closing database connection
<code>imq.persist.jdbc.user</code>		<p>(Optional) User name for opening database connection, if required</p> <p>For security reasons, the value can instead be specified using command line options <code>imqbrokerd -dbuser</code> and <code>imqdbmgr -u</code>.</p>

TABLE 14–6 Broker Properties for JDBC-Based Persistence (Continued)

Property	Example	Description
<code>imq.persist.jdbc.needpassword</code>		(Optional) Does database require a password for broker access? If <code>true</code> , the <code>imqbrokerd</code> and <code>imqdbmgr</code> commands will prompt for a password, unless you use the <code>-passfile</code> option to specify a password file containing it.
<code>imq.persist.jdbc.password¹</code>		(Optional) Password for opening database connection
<code>imq.persist.jdbc.table.IMQSV35</code>	CREATE TABLE <code>\${name}</code> (STOREVERSION INTEGER NOT NULL, BROKERID VARCHAR(100))	SQL command to create version table
<code>imq.persist.jdbc.table.IMQCCREC35</code>	CREATE TABLE <code>\${name}</code> (RECORDTIME BIGINT NOT NULL, RECORD BLOB(10k))	SQL command to create configuration change record table
<code>imq.persist.jdbc.table.IMQDEST35</code>	CREATE TABLE <code>\${name}</code> (DID VARCHAR(100) NOT NULL, DEST BLOB(10k), <i>primaryKey</i> (DID))	SQL command to create destination table
<code>imq.persist.jdbc.table.IMQINT35</code>	CREATE TABLE <code>\${name}</code> (CUID BIGINT NOT NULL, INTEREST BLOB(10k), <i>primaryKey</i> (CUID))	SQL command to create interest table
<code>imq.persist.jdbc.table.IMQMSG35</code>	CREATE TABLE <code>\${name}</code> (MID VARCHAR(100) NOT NULL, DID VARCHAR(100), MSGSIZE BIGINT, MSG BLOB(1m), <i>primaryKey</i> (MID))	SQL command to create message table The default maximum length for the MSG column is 1 megabyte (1m). If you expect to have messages larger than this, set the length accordingly. If the tables have already been created, you must re-create them to change the maximum message length.
<code>imq.persist.jdbc.table.IMQPROPS35</code>	CREATE TABLE <code>\${name}</code> (PROPNAME VARCHAR(100) NOT NULL, PROPVALUE BLOB(10k), <i>primaryKey</i> (PROPNAME))	SQL command to create property table

¹ Should be used only in password files

TABLE 14–6 Broker Properties for JDBC-Based Persistence (Continued)

Property	Example	Description
imq.persist.jdbc.table.IMQILIST35	CREATE TABLE \${name} (MID VARCHAR(100) NOT NULL, CUID BIGINT, DID VARCHAR(100), STATE INTEGER, primaryKey(MID, CUID))	SQL command to create interest state table
imq.persist.jdbc.table.IMQTXN35	CREATE TABLE \${name} (TUID BIGINT NOT NULL, STATE INTEGER, TSTATEOBJ BLOB(10K), primaryKey(TUID))	SQL command to create transaction table
imq.persist.jdbc.table.IMQTACK35	CREATE TABLE \${name} (TUID BIGINT NOT NULL, TXNACK BLOB(10k))	SQL command to create transaction acknowledgment table

Security Properties

Table 14–7 lists the broker properties related to security services.

TABLE 14–7 Broker Security Properties

Property	Type	Default	Description
imq.accesscontrol.enabled	Boolean	true	Use access control? If true, the system will check the access control properties file to verify that an authenticated user is authorized to use a connection service or to perform specific operations with respect to specific destinations.

TABLE 14-7 Broker Security Properties (Continued)

Property	Type	Default	Description
<code>imq.serviceName.accesscontrol.enabled</code>	Boolean	None	<p>Use access control for connection service?</p> <p>If specified, overrides <code>imq.accesscontrol.enabled</code> for the designated connection service.</p> <p>If <code>true</code>, the system will check the access control properties file to verify that an authenticated user is authorized to use the designated connection service or to perform specific operations with respect to specific destinations.</p>
<code>imq.accesscontrol.file.filename</code>	String	<code>accesscontrol.properties</code>	<p>Name of access control properties file</p> <p>The file name specifies a path relative to the access control directory (see Appendix A).</p>
<code>imq.serviceName.accesscontrol.file.filename</code>	String	None	<p>Name of access control properties file for connection service</p> <p>If specified, overrides <code>imq.accesscontrol.file.filename</code> for the designated connection service.</p> <p>The file name specifies a path relative to the access control directory (see Appendix A).</p>
<code>imq.authentication.type</code>	String	<code>digest</code>	<p>Password encoding method:</p> <ul style="list-style-type: none"> <code>basic</code>: Base-64 <code>digest</code>: MD5
<code>imq.serviceName.authentication.type</code>	String	None	<p>Password encoding method for connection service:</p> <ul style="list-style-type: none"> <code>basic</code>: Base-64 <code>digest</code>: MD5 <p>If specified, overrides <code>imq.authentication.type</code> for the designated connection service.</p>

TABLE 14-7 Broker Security Properties (Continued)

Property	Type	Default	Description
<code>imq.authentication.basic.user_repository</code>	String	<code>file</code>	Type of user repository for base-64 authentication: <code>file</code> : File-based <code>ldap</code> : LDAP
<code>imq.authentication.client.response.timeout</code>	Integer	<code>180</code>	Interval, in seconds, to wait for client response to authentication requests
<code>imq.passfile.enabled</code>	Boolean	<code>false</code>	Obtain passwords from password file?
<code>imq.passfile.dirpath</code>	String	See Appendix A	Path to directory containing password file
<code>imq.passfile.name</code>	String	<code>passfile</code>	Name of password file
<code>imq.imqcmd.password</code>	String	<code>None</code>	Password for administrative user The Command utility (<code>imqcmd</code>) uses this password to authenticate the user before executing a command.
<code>imq.user_repository.ldap.server</code>	String	<code>None</code>	Host name and port number for LDAP server The value is of the form <code>hostName:port</code> where <i>hostName</i> is the fully qualified DNS name of the host running the LDAP server and <i>port</i> is the port number used by the server. To specify a list of failover servers, use the following syntax: <code>host1:port1</code> <code>ldap://host2:port2</code> <code>ldap://host3:port3</code> ...

TABLE 14-7 Broker Security Properties (Continued)

Property	Type	Default	Description
			Entries in the list are separated by spaces. Note that each failover server address is prefixed with <code>ldap://</code> . Use this format even if you use SSL and have set the property <code>imq.user_repository.ldap.ssl.enabled</code> to <code>true</code> . You need not specify <code>ldaps</code> in the address.
<code>imq.user_repository.ldap.principal</code>	String	None	Distinguished name for binding to LDAP user repository Not needed if the LDAP server allows anonymous searches.
<code>imq.user_repository.ldap.password¹</code>	String	None	Password for binding to LDAP user repository Not needed if the LDAP server allows anonymous searches.
<code>imq.user_repository.ldap.propertyName</code>	To come	To come	To come
<code>imq.user_repository.ldap.base</code>	String	None	Directory base for LDAP user entries
<code>imq.user_repository.ldap.uidattr</code>	String	None	Provider-specific attribute identifier for LDAP user name
<code>imq.user_repository.ldap.usrfilter</code>	String	None	(Optional) JNDI filter for LDAP user searches
<code>imq.user_repository.ldap.grpsearch</code>	Boolean	false	Enable LDAP group searches? Note – Message Queue does not support nested groups.
<code>imq.user_repository.ldap.grpbase</code>	String	None	Directory base for LDAP group entries
<code>imq.user_repository.ldap.gidattr</code>	String	None	Provider-specific attribute identifier for LDAP group name
<code>imq.user_repository.ldap.memattr</code>	String	None	Provider-specific attribute identifier for user names in LDAP group
<code>imq.user_repository.ldap.grpfilter</code>	String	None	(Optional) JNDI filter for LDAP group searches

¹ Should be used only in password files

TABLE 14–7 Broker Security Properties (Continued)

Property	Type	Default	Description
imq.user_repository.ldap.timeout	Integer	280	Time limit for LDAP searches, in seconds
imq.user_repository.ldap.ssl.enabled	Boolean	false	Use SSL when communicating with LDAP server?
imq.keystore.file.dirpath	String	See Appendix A	Path to directory containing key store file
imq.keystore.file.name	String	keystore	Name of key store file
imq.keystore.password ¹	String	None	Password for key store file
imq.audit.enabled ²	Boolean	false	Start audit logging to broker log file?

¹ Should be used only in password files

² Message Queue Enterprise Edition only

Monitoring Properties

[Table 14–8](#) lists the broker properties related to monitoring services.

TABLE 14–8 Broker Monitoring Properties

Property	Type	Default	Description
imq.log.level ¹	String	INFO	Logging level Specifies the categories of logging information that can be written to an output channel. Possible values, from high to low: <div>ERROR</div> <div>WARNING</div> <div>INFO</div> Each level includes those above it (for example, WARNING includes ERROR).

¹ Can be used with imqcmd update bkr command

TABLE 14–8 Broker Monitoring Properties (Continued)

Property	Type	Default	Description
<code>imq.destination.logDeadMsgs</code> ¹	Boolean	false	Log information about dead messages? If <code>true</code> , the following events will be logged: <ul style="list-style-type: none"> ■ A destination is full, having reached its maximum size or message count. ■ The broker discards a message for a reason other than an administrative command or delivery acknowledgment. ■ The broker moves a message to the dead message queue.
<code>imq.log.console.stream</code>	String	ERR	Destination for console output: OUT: <code>stdout</code> ERR: <code>stderr</code>
<code>imq.log.console.output</code>	String	ERROR WARNING	Categories of logging information to write to console: NONE ERROR WARNING INFO ALL The ERROR, WARNING, and INFO categories do <i>not</i> include those above them, so each must be specified explicitly if desired. Any combination of categories can be specified, separated by vertical bars (<code> </code>).
<code>imq.log.file.dirpath</code>	String	See Appendix A	Path to directory containing log file
<code>imq.log.file.filename</code>	String	<code>log.txt</code>	Name of log file

¹ Can be used with `imqcmd update bkr` command

TABLE 14–8 Broker Monitoring Properties (Continued)

Property	Type	Default	Description
<code>imq.log.file.output</code>	String	ALL	<p>Categories of logging information to write to log file:</p> <p>NONE ERROR WARNING INFO ALL</p> <p>The ERROR, WARNING, and INFO categories do <i>not</i> include those above them, so each must be specified explicitly if desired. Any combination of categories can be specified, separated by vertical bars ().</p>
<code>imq.log.file.rolloverbytes</code> ¹	Integer	-1	<p>File length, in bytes, at which output rolls over to a new log file</p> <p>A value of -1 denotes an unlimited number of bytes (no rollover based on file length).</p>
<code>imq.log.file.rolloversecs</code> ¹	Integer	604800 (one week)	<p>Age of file, in seconds, at which output rolls over to a new log file</p> <p>A value of -1 denotes an unlimited number of seconds (no rollover based on file age).</p>
<code>imq.log.syslog.output</code> ²	String	ERROR	<p>Categories of logging information to write to <code>syslogd(1M)</code>:</p> <p>NONE ERROR WARNING INFO ALL</p> <p>The ERROR, WARNING, and INFO categories do <i>not</i> include those above them, so each must be specified explicitly if desired. Any combination of categories can be specified, separated by vertical bars ().</p>

¹ Can be used with `imqcmd update bkr` command² Solaris platform only

TABLE 14–8 Broker Monitoring Properties (Continued)

Property	Type	Default	Description
<code>imq.log.syslog.facility²</code>	String	<code>LOG_DAEMON</code>	<p>syslog facility for logging messages</p> <p>Possible values mirror those listed on the <code>syslog(3C)</code> man page. Appropriate values for use with Message Queue include:</p> <ul style="list-style-type: none"> <code>LOG_USER</code> <code>LOG_DAEMON</code> <code>LOG_LOCAL0</code> <code>LOG_LOCAL1</code> <code>LOG_LOCAL2</code> <code>LOG_LOCAL3</code> <code>LOG_LOCAL4</code> <code>LOG_LOCAL5</code> <code>LOG_LOCAL6</code> <code>LOG_LOCAL7</code>
<code>imq.log.syslog.identity²</code>	String	<code>imqbrokerd_\${imq.instanceName}</code>	Identity string to be prefixed to all messages logged to syslog
<code>imq.log.syslog.logpid²</code>	Boolean	<code>true</code>	Log broker process ID with message?
<code>imq.log.syslog.logconsole²</code>	Boolean	<code>false</code>	Write messages to system console if they cannot be sent to syslog?
<code>imq.log.timezone</code>	String	Local time zone	<p>Time zone for log time stamps</p> <p>Possible values are the same as those used by the method <code>java.util.TimeZone.getTimeZone</code>.</p> <p>Examples:</p> <ul style="list-style-type: none"> <code>GMT</code> <code>GMT-8:00</code> <code>America/LosAngeles</code> <code>Europe/Rome</code> <code>Asia/Tokyo</code>
<code>imq.metrics.enabled</code>	Boolean	<code>true</code>	<p>Enable writing of metrics information to Logger?</p> <p>Does not affect the production of metrics messages (controlled by <code>imq.metrics.topic.enabled</code>).</p>

² Solaris platform only

TABLE 14–8 Broker Monitoring Properties (Continued)

Property	Type	Default	Description
imq.metrics.interval	Integer	-1	Time interval, in seconds, at which to write metrics information to Logger Does not affect the time interval for production of metrics messages (controlled by <code>imq.metrics.topic.interval</code>). A value of -1 denotes an indefinite interval (never write metrics information to Logger).
imq.metrics.topic.enabled	Boolean	true	Enable production of metrics messages to metric topic destinations? If false, an attempt to subscribe to a metric topic destination will throw a client-side exception.
imq.metrics.topic.interval	Integer	60	Time interval, in seconds, at which to produce metrics messages to metric topic destinations
imq.metrics.topic.persist	Boolean	false	Are metrics messages sent to metric topic destinations persistent?
imq.metrics.topic.timetolive	Integer	300	Lifetime, in seconds, of metrics messages sent to metric topic destinations

Cluster Configuration Properties

Table 14–9 lists the configuration properties related to broker clusters.

TABLE 14–9 Broker Properties for Cluster Configuration

Property	Type	Default	Description
imq.cluster.brokerlist ¹	String	None	List of broker addresses The list consists of one or more addresses, separated by commas. Each address specifies the host name and Port Mapper port number of a broker in the cluster, in the form <i>hostName:portNumber</i> . Example: host1:3000,host2:8000,ctrlhost

¹ Must have the same value for all brokers in a cluster

TABLE 14–9 Broker Properties for Cluster Configuration (Continued)

Property	Type	Default	Description
<code>imq.cluster.hostname²</code>	String	None	Host name or IP address for cluster connection service If specified, overrides <code>imq.hostname</code> (see Table 14–1) for the cluster connection service.
<code>imq.cluster.port²</code>	Integer	0	Port number for cluster connection service A value of 0 specifies that the port number should be allocated dynamically by the Port Mapper.
<code>imq.cluster.transport¹</code>	String	tcp	Network transport protocol for cluster connection service For secure, encrypted message delivery between brokers, set this property to <code>ssl</code> .
<code>imq.cluster.url^{1,3}</code>	String	None	URL of cluster configuration file, if any Examples: <code>http://webserver/imq/cluster.properties</code> (for a file on a Web server) <code>file:/net/mfsserver/imq/cluster.properties</code> (for a file on a shared drive)
<code>imq.cluster.masterbroker¹</code>	String	None	Host name and port number of cluster's master broker, if any The value has the form <code>hostName:portNumber</code> , where <code>hostName</code> is the host name of the master broker and <code>portNumber</code> is its Port Mapper port number. Example: <code>ctrlhost:7676</code>

² Can be specified independently for each broker in a cluster¹ Must have the same value for all brokers in a cluster³ Can be used with `imqcmd update bkr` command

Alphabetical List of Broker Properties

“[Alphabetical List of Broker Properties](#)” on page 283 is an alphabetical list of broker configuration properties, with cross-references to the relevant tables in this chapter.

TABLE 14-10 Alphabetical List of Broker Properties

Property	Table
<code>imq.accesscontrol.enabled</code>	Table 14-7
<code>imq.accesscontrol.file.filename</code>	Table 14-7
<code>imq.audit.enabled</code>	Table 14-7
<code>imq.authentication.basic.user_repository</code>	Table 14-7
<code>imq.authentication.client.response.timeout</code>	Table 14-7
<code>imq.authentication.type</code>	Table 14-7
<code>imq.autocreate.destination.isLocalOnly</code>	Table 14-3
<code>imq.autocreate.destination.limitBehavior</code>	Table 14-3
<code>imq.autocreate.destination.maxBytesPerMsg</code>	Table 14-3
<code>imq.autocreate.destination.maxNumMsgs</code>	Table 14-3
<code>imq.autocreate.destination.maxNumProducers</code>	Table 14-3
<code>imq.autocreate.destination.maxTotalMsgBytes</code>	Table 14-3
<code>imq.autocreate.destination.useDMQ</code>	Table 14-3
<code>imq.autocreate.queue</code>	Table 14-3
<code>imq.autocreate.queue.consumerFlowLimit</code>	Table 14-3
<code>imq.autocreate.queue.localDeliveryPreferred</code>	Table 14-3
<code>imq.autocreate.queue.maxNumActiveConsumers</code>	Table 14-3
<code>imq.autocreate.queue.maxNumBackupConsumers</code>	Table 14-3
<code>imq.autocreate.topic</code>	Table 14-3
<code>imq.autocreate.topic.consumerFlowLimit</code>	Table 14-3
<code>imq.cluster.brokerlist</code>	Table 14-9
<code>imq.cluster.hostname</code>	Table 14-9
<code>imq.cluster.masterbroker</code>	Table 14-9
<code>imq.cluster.port</code>	Table 14-9
<code>imq.cluster.transport</code>	Table 14-9
<code>imq.cluster.url</code>	Table 14-9
<code>imq.destination.DMQ.truncateBody</code>	Table 14-2

TABLE 14–10 Alphabetical List of Broker Properties (Continued)

Property	Table
<code>imq.destination.logDeadMsgs</code>	Table 14–8
<code>imq.hostname</code>	Table 14–1
<code>imq.imqcmd.password</code>	Table 14–7
<code>imq.keystore.file.dirpath</code>	Table 14–7
<code>imq.keystore.file.name</code>	Table 14–7
<code>imq.keystore.password</code>	Table 14–7
<code>imq.keystore.propertyName</code>	Table 14–7
<code>imq.log.console.output</code>	Table 14–8
<code>imq.log.console.stream</code>	Table 14–8
<code>imq.log.file.dirpath</code>	Table 14–8
<code>imq.log.file.filename</code>	Table 14–8
<code>imq.log.file.output</code>	Table 14–8
<code>imq.log.file.rolloverbytes</code>	Table 14–8
<code>imq.log.file.rolloversecs</code>	Table 14–8
<code>imq.log.level</code>	Table 14–8
<code>imq.log.syslog.facility</code>	Table 14–8
<code>imq.log.syslog.identity</code>	Table 14–8
<code>imq.log.syslog.logconsole</code>	Table 14–8
<code>imq.log.syslog.logpid</code>	Table 14–8
<code>imq.log.syslog.output</code>	Table 14–8
<code>imq.log.timezone</code>	Table 14–8
<code>imq.message.expiration.interval</code>	Table 14–2
<code>imq.message.max_size</code>	Table 14–2
<code>imq.metrics.enabled</code>	Table 14–8
<code>imq.metrics.interval</code>	Table 14–8
<code>imq.metrics.topic.enabled</code>	Table 14–8
<code>imq.metrics.topic.interval</code>	Table 14–8
<code>imq.metrics.topic.persist</code>	Table 14–8

TABLE 14-10 Alphabetical List of Broker Properties (Continued)

Property	Table
imq.metrics.topic.timetolive	Table 14-8
imq.passfile.dirpath	Table 14-7
imq.passfile.enabled	Table 14-7
imq.passfile.name	Table 14-7
imq.persist.file.destination.message.filepool.limit	Table 14-5
imq.persist.file.message.cleanup	Table 14-5
imq.persist.file.message.filepool.cleanratio	Table 14-5
imq.persist.file.message.max_record_size	Table 14-5
imq.persist.file.sync.enabled	Table 14-5
imq.persist.jdbc.brokerid	Table 14-6
imq.persist.jdbc.closedburl	Table 14-6
imq.persist.jdbc.createdburl	Table 14-6
imq.persist.jdbc.driver	Table 14-6
imq.persist.jdbc.needpassword	Table 14-6
imq.persist.jdbc.opendburl	Table 14-6
imq.persist.jdbc.password	Table 14-6
imq.persist.jdbc.table.IMQCCREC35	Table 14-6
imq.persist.jdbc.table.IMQDEST35	Table 14-6
imq.persist.jdbc.table.IMQILIST35	Table 14-6
imq.persist.jdbc.table.IMQINT35	Table 14-6
imq.persist.jdbc.table.IMQMSG35	Table 14-6
imq.persist.jdbc.table.IMQPROPS35	Table 14-6
imq.persist.jdbc.table.IMQSV35	Table 14-6
imq.persist.jdbc.table.IMQTACK35	Table 14-6
imq.persist.jdbc.table.IMQTXN35	Table 14-6
imq.persist.jdbc.user	Table 14-6
imq.persist.store	Table 14-4
imq.ping.interval	Table 14-1

TABLE 14–10 Alphabetical List of Broker Properties (Continued)

Property	Table
<code>imq.portmapper.backlog</code>	Table 14–1
<code>imq.portmapper.hostname</code>	Table 14–1
<code>imq.portmapper.port</code>	Table 14–1
<code>imq.resourceState.count</code>	Table 14–2
<code>imq.resourceState.threshold</code>	Table 14–2
<code>imq.service.activelist</code>	Table 14–1
<code>imq.serviceName.accesscontrol.enabled</code>	Table 14–7
<code>imq.serviceName.accesscontrol.file.filename</code>	Table 14–7
<code>imq.serviceName.authentication.type</code>	Table 14–7
<code>imq.serviceName.max_threads</code>	Table 14–1
<code>imq.serviceName.min_threads</code>	Table 14–1
<code>imq.serviceName.protocolType.hostname</code>	Table 14–1
<code>imq.serviceName.protocolType.port</code>	Table 14–1
<code>imq.serviceName.threadpool_model</code>	Table 14–1
<code>imq.shared.connectionMonitor_limit</code>	Table 14–1
<code>imq.system.max_count</code>	Table 14–2
<code>imq.system.max_size</code>	Table 14–2
<code>imq.transaction.autorollback</code>	Table 14–2
<code>imq.user_repository.ldap.base</code>	Table 14–7
<code>imq.user_repository.ldap.gidattr</code>	Table 14–7
<code>imq.user_repository.ldap.grpbases</code>	Table 14–7
<code>imq.user_repository.ldap.grpfilter</code>	Table 14–7
<code>imq.user_repository.ldap.grpsearch</code>	Table 14–7
<code>imq.user_repository.ldap.memattr</code>	Table 14–7
<code>imq.user_repository.ldap.password</code>	Table 14–7
<code>imq.user_repository.ldap.principal</code>	Table 14–7
<code>imq.user_repository.ldap.propertyName</code>	Table 14–7
<code>imq.user_repository.ldap.server</code>	Table 14–7

TABLE 14–10 Alphabetical List of Broker Properties *(Continued)*

Property	Table
imq.user_repository.ldap.ssl.enabled	Table 14–7
imq.user_repository.ldap.timeout	Table 14–7
imq.user_repository.ldap.uidattr	Table 14–7
imq.user_repository.ldap.usrfilter	Table 14–7

Physical Destination Property Reference

This chapter provides reference information about configuration properties for physical destinations. These properties can be set when creating or updating a physical destination. For auto-created destinations, you set default values in the broker's instance configuration file (see [Table 14-3](#)).

Physical Destination Properties

TABLE 15-1 Physical Destination Properties

Property	Type	Default	Description
maxNumMsgs ¹	Integer	- 1	Maximum number of unconsumed messages A value of - 1 denotes an unlimited number of messages. For the dead message queue, the default value is 1000.
maxBytesPerMsg	String	- 1	Maximum size, in bytes, of any single message Rejection of a persistent message is reported to the producing client with an exception; no notification is sent for nonpersistent messages. The value may be expressed in bytes, kilobytes, or megabytes, using the following suffixes: b: Bytes k: Kilobytes (1024 bytes) m: Megabytes (1024 × 1024 = 1,048,576 bytes) An unsuffixed value is expressed in bytes; a value of - 1 denotes an unlimited message size.

¹ In a cluster environment, applies to each individual instance of a destination rather than collectively to all instances in the cluster

TABLE 15-1 Physical Destination Properties (Continued)

Property	Type	Default	Description
			Examples: 1600: 1600 bytes 1600b: 1600 bytes 16k: 16 kilobytes (= 16,384 bytes) 16m: 16 megabytes (= 16,777,216 bytes) -1: No limit
maxTotalMsgBytes ¹	String	-1	Maximum total memory, in bytes, for unconsumed messages The syntax is the same as for maxBytesPerMsg (see above). For the dead message queue, the default value is 10m.
limitBehavior	String	REJECT_NEWEST	Broker behavior when memory-limit threshold reached: FLOW_CONTROL: Slow down producers REMOVE_OLDEST: Throw out oldest messages REMOVE_LOW_PRIORITY: Throw out lowest-priority messages according to age; no notification to producing client REJECT_NEWEST: Reject newest messages; notify producing client with an exception only if message is persistent If the value is REMOVE_OLDEST or REMOVE_LOW_PRIORITY and the useDMQ property is true, excess messages are moved to the dead message queue. For the dead message queue itself, the default limit behavior is REMOVE_OLDEST and cannot be set to FLOW_CONTROL.
maxNumProducers ²	Integer	-1	Maximum number of message producers for destination When this limit is reached, no new producers can be created. A value of -1 denotes an unlimited number of producers.
maxNumActiveConsumers ³	Integer	1	Maximum number of active message consumers in load-balanced delivery from queue destination A value of -1 denotes an unlimited number of consumers. In Sun Java System Message Queue™ Platform Edition, the value is limited to 2.

¹ In a cluster environment, applies to each individual instance of a destination rather than collectively to all instances in the cluster² Cannot be set for dead message queue³ Queue destinations only.

TABLE 15-1 Physical Destination Properties (Continued)

Property	Type	Default	Description
maxNumBackupConsumers ³	Integer	0	Maximum number of backup message consumers in load-balanced delivery from queue destination A value of -1 denotes an unlimited number of consumers. In Sun Java System Message Queue Platform Edition, the value is limited to 1.
consumerFlowLimit	Integer	1000	Maximum number of messages delivered to consumer in a single batch In load-balanced queue delivery, this is the initial number of queued messages routed to active consumers before load balancing begins. A destination consumer can override this limit by specifying a lower value on a connection. A value of -1 denotes an unlimited number of messages.
isLocalOnly ²	Boolean	false	Local delivery only? This property applies only to destinations in broker clusters, and cannot be changed once the destination has been created. If true, the destination is not replicated on other brokers and is limited to delivering messages only to local consumers (those connected to the broker on which the destination is created).
localDeliveryPreferred ^{2,3}	Boolean	false	Local delivery preferred? This property applies only to load-balanced queue delivery in broker clusters. If true, messages will be delivered to remote consumers only if there are no consumers on the local broker; the destination must not be restricted to local-only delivery (isLocalOnly must be false).
useDMQ ²	Boolean	true	Send dead messages to dead message queue? If false, dead messages will simply be discarded.

³ Queue destinations only.² Cannot be set for dead message queue

Administered Object Attribute Reference

This chapter provides reference information about the attributes of administered objects. It consists of the following sections:

- [“Connection Factory Attributes” on page 293](#)
- [“Destination Attributes” on page 300](#)
- [“SOAP Endpoint Attributes” on page 301](#)

Connection Factory Attributes

The attributes of a connection factory object are grouped into categories described in the following sections below:

- [“Connection Handling” on page 293](#)
- [“Client Identification” on page 297](#)
- [“Reliability and Flow Control” on page 297](#)
- [“Queue Browser and Server Sessions” on page 298](#)
- [“Standard Message Properties” on page 299](#)
- [“Message Header Overrides” on page 300](#)

Connection Handling

[Table 16–1](#) lists the connection factory attributes for connection handling.

TABLE 16-1 Connection Factory Attributes for Connection Handling

Attribute	Type	Default	Description
imqAddressList	String	An existing Message Queue™ 3.0 address, if any; if none, the first entry in Table 16-2	<p>List of broker addresses</p> <p>The list consists of one or more broker addresses, separated by commas. Each address specifies (or implies) the host name, port number, and connection service for a broker instance to which the client can connect. Address syntax varies depending on the connection service and port assignment method; see below for details.</p>
imqAddressListBehavior	String	PRIORITY	<p>Order in which to attempt connection to broker addresses:</p> <p>PRIORITY: Order specified in address list</p> <p>RANDOM: Random order</p> <p>Note – If many clients share the same connection factory, specify random connection order to prevent them from all attempting to connect to the same address.</p>
imqAddressListIterations	Integer	5	<p>Number of times to iterate through address list attempting to establish or reestablish a connection</p> <p>A value of -1 denotes an unlimited number of iterations.</p>
imqPingInterval	Integer	30	<p>Interval, in seconds, at which to test connection between client and broker</p> <p>A value of 0 or -1 disables periodic testing of the connection.</p>
imqReconnectEnabled	Boolean	false	Attempt to reestablish a lost connection?
imqReconnectAttempts	Integer	0	<p>Number of times to attempt connection (or reconnection) to each address in address list before moving on to next</p> <p>A value of -1 denotes an unlimited number of connection attempts: attempt repeatedly to connect to first address until successful.</p>
imqReconnectInterval	Long integer	3000	<p>Interval, in milliseconds, between reconnection attempts</p> <p>This value applies both for successive attempts on a given address and for successive addresses in the list.</p> <p>Note – Too small a value may give the broker insufficient recovery time; too large a value may cause unacceptable connection delays.</p>

TABLE 16–1 Connection Factory Attributes for Connection Handling (Continued)

Attribute	Type	Default	Description
imqSSLIsHostTrusted	Boolean	true	<p>Trust any certificate presented by broker?</p> <p>If false, the Message Queue client runtime will validate all certificates presented to it. Validation will fail if the signer of the certificate is not in the client's trust store.</p> <p>If true, validation of certificates is skipped. This can be useful, for instance, during software testing when a self-signed certificate is used.</p> <p>NOTE: To use signed certificates from a certification authority, set this attribute to false.</p>

The value of the `imqAddressList` attribute is a comma-separated string specifying one or more broker addresses to which to connect. The general syntax for each address is as follows:

scheme://*address*

where *scheme* identifies one of the addressing schemes shown in the first column of [Table 16–2](#) and *address* denotes the broker address itself. The exact syntax for specifying the address depends on the addressing scheme, as shown in the last column of the table.

TABLE 16–2 Message Broker Addressing Schemes

Scheme	Service	Syntax	Description
mq	jms or ssljms	[<i>hostName</i>][: <i>portNumber</i>]/[<i>serviceName</i>]	<p>Assign port dynamically for <code>jms</code> or <code>ssljms</code> connection service</p> <p>The address list entry specifies the host name and port number for the Message Queue Port Mapper. The Port Mapper itself dynamically assigns a port to be used for the connection.</p> <p>Default values:</p> <p><i>hostName</i> = localhost</p> <p><i>portNumber</i> = 7676</p> <p><i>serviceName</i> = <code>jms</code></p> <p>For the <code>ssljms</code> connection service, all variables must be specified explicitly.</p>
mqtcp	jms	<i>hostName</i> : <i>portNumber</i> / <i>jms</i>	<p>Connect to specified port using <code>jms</code> connection service</p> <p>Bypasses the Port Mapper and makes a TCP connection directly to the specified host name and port number.</p>

TABLE 16-2 Message Broker Addressing Schemes (Continued)

Scheme	Service	Syntax	Description
mqssl	ssljms	<i>hostName:portNumber/ssljms</i>	Connect to specified port using ssljms connection service Bypasses the Port Mapper and makes a secure SSL connection directly to the specified host name and port number.
http	httpjms	<i>http://hostName:portNumber/contextRoot/tunnel</i> If multiple broker instances use the same tunnel servlet, the following syntax connects to a specific broker instance rather than a randomly selected one: <i>http://hostName:portNumber/contextRoot/tunnel?ServerName=hostName:instanceName</i>	Connect to specified port using httpjms connection service Makes an HTTP connection to a Message Queue tunnel servlet at the specified URL. The broker must be configured to access the HTTP tunnel servlet.
https	httpsjms	<i>https://hostName:portNumber/contextRoot/tunnel</i> If multiple broker instances use the same tunnel servlet, the following syntax connects to a specific broker instance rather than a randomly selected one: <i>https://hostName:portNumber/contextRoot/tunnel?ServerName=hostName:instanceName</i>	Connect to specified port using httpsjms connection service Makes a secure HTTPS connection to a Message Queue tunnel servlet at the specified URL. The broker must be configured to access the HTTPS tunnel servlet.

TABLE 16-3 Message Broker Address Examples

Service	Broker Host	Port	Example Address
Not specified	Not specified	Not specified	No address (mq://localhost:7676/jms)
Not specified	Specified host	Not specified	myBkrHost (mq://myBkrHost:7676/jms)
Not specified	Not specified	Specified Port Mapper port	1012 (mq://localhost:1012/jms)
ssljms	Local host	Standard Port Mapper port	mq://localhost:7676/ssljms
ssljms	Specified host	Standard Port Mapper port	mq://myBkrHost:7676/ssljms
ssljms	Specified host	Specified Port Mapper port	mq://myBkrHost:1012/ssljms
jms	Local host	Specified service port	mqtcp://localhost:1032/jms
ssljms	Specified host	Specified service port	mqssl://myBkrHost:1034/ssljms
httpjms	Not applicable	Not applicable	http://websrvr1:8085/imq/tunnel
httpsjms	Not applicable	Not applicable	https://websrvr2:8090/imq/tunnel

Client Identification

Table 16–4 lists the connection factory attributes for client identification.

TABLE 16–4 Connection Factory Attributes for Client Identification

Attribute	Type	Default	Description
imqDefaultUsername	String	guest	Default user name for authenticating with broker
imqDefaultPassword	String	guest	Default password for authenticating with broker
imqConfiguredClientID	String	null	Administratively configured client identifier
imqDisableSetClientID	Boolean	false	Prevent client from changing client identifier using setClientID method?

Reliability and Flow Control

Table 16–5 lists the connection factory attributes for reliability and flow control.

TABLE 16–5 Connection Factory Attributes for Reliability and Flow Control

Attribute	Type	Default	Description
imqAckTimeout	String	0	<p>Maximum time, in milliseconds, to wait for broker acknowledgment before throwing an exception</p> <p>A value of 0 denotes no timeout (wait indefinitely).</p> <p>Note – In some situations, too low a value can cause premature timeout: for example, initial authentication of a user against an LDAP user repository using a secure (SSL) connection can take more than 30 seconds.</p>
imqConnectionFlowCount	Integer	100	<p>Number of payload messages in a metered batch</p> <p>Delivery of payload messages to the client is temporarily suspended after this number of messages, allowing any accumulated control messages to be delivered. Payload message delivery is resumed on notification by the client runtime, and continues until the count is again reached.</p> <p>A value of 0 disables metering of message delivery and may cause Message Queue control messages to be blocked by heavy payload message traffic.</p>
imqConnectionFlowLimitEnabled	Boolean	false	Limit message flow at connection level?

TABLE 16–5 Connection Factory Attributes for Reliability and Flow Control (Continued)

Attribute	Type	Default	Description
imqConnectionFactoryLimit	Integer	1000	<p>Maximum number of messages per connection to deliver and buffer for consumption</p> <p>Message delivery on a connection stops when the number of unconsumed payload messages pending (subject to flow metering governed by <code>imqConnectionFactoryCount</code>) exceeds this limit. Delivery resumes only when the number of pending messages falls below the limit. This prevents the client from being overwhelmed with pending messages that might cause it to run out of memory.</p> <p>This attribute is ignored if <code>imqConnectionFactoryLimitEnabled</code> is <code>false</code>.</p>
imqConsumerFlowLimit	Integer	100	<p>Maximum number of messages per consumer to deliver and buffer for consumption</p> <p>Message delivery to a given consumer stops when the number of unconsumed payload messages pending for that consumer exceeds this limit. Delivery resumes only when the number of pending messages for the consumer falls below the percentage specified by <code>imqConsumerFlowThreshold</code>. This can be used to improve load balancing among multiple consumers and prevent any single consumer from starving others on the same connection.</p> <p>This limit can be overridden by a lower value set for a queue's own <code>consumerFlowLimit</code> attribute (see Chapter 15). Note also that message delivery to all consumers on a connection is subject to the overall limit specified by <code>imqConnectionFactoryLimit</code>.</p>
imqConsumerFlowThreshold	Integer	50	<p>Number of messages per consumer buffered in the client runtime, as a percentage of <code>imqConsumerFlowLimit</code>, below which to resume message delivery</p>

Queue Browser and Server Sessions

[Table 16–6](#) lists the connection factory attributes for queue browsing and server sessions.

TABLE 16–6 Connection Factory Attributes for Queue Browser and Server Sessions

Attribute	Type	Default	Description
imqQueueBrowserMaxMessagesPerRetrieve	Integer	1000	<p>Maximum number of messages to retrieve at one time when browsing contents of a queue destination</p> <p>Note – This attribute does not affect the total number of messages browsed, only the way they are chunked for delivery to the client runtime (fewer but larger chunks or more but smaller ones). The client application will always receive all messages in the queue. Changing the attribute's value may affect performance, but will not affect the total amount of data retrieved.</p>
imqQueueBrowserRetrieveTimeout	Long integer	60000	Maximum time, in milliseconds, to wait to retrieve messages, when browsing contents of a queue destination, before throwing an exception
imqLoadMaxToServerSession	Boolean	true	<p>Load up to maximum number of messages into a server session?</p> <p>If false, the client will load only a single message at a time.</p> <p>This attribute applies only to JMS application server facilities.</p>

Standard Message Properties

The connection factory attributes listed in [Table 16–7](#) control whether the Message Queue client runtime sets certain standard message properties defined in the *Java Message Service Specification*.

TABLE 16–7 Connection Factory Attributes for Standard Message Properties

Property	Type	Default	Description
imqSetJMSXUserID	Boolean	false	Set JMSXUserID property (identity of user sending message) for produced messages?
imqSetJMSXAppID	Boolean	false	Set JMSXAppID property (identity of application sending message) for produced messages?
imqSetJMSXProducerTXID	Boolean	false	Set JMSXProducerTXID property (transaction identifier of transaction within which message was produced) for produced messages?

TABLE 16–7 Connection Factory Attributes for Standard Message Properties (Continued)

Property	Type	Default	Description
imqSetJMSXConsumerTXID	Boolean	false	Set JMSXConsumerTXID property (transaction identifier of transaction within which message was consumed) for consumed messages?
imqSetJMSXRcvTimestamp	Boolean	false	Set JMSXRcvTimestamp property (time message delivered to consumer) for consumed messages?

Message Header Overrides

Table 16–8 lists the connection factory attributes for overriding JMS message header fields.

TABLE 16–8 Connection Factory Attributes for Message Header Overrides

Attribute	Type	Default	Description
imqOverrideJMSDeliveryMode	Boolean	false	Allow client-set delivery mode to be overridden?
imqJMSDeliveryMode	Integer	2	Overriding value of delivery mode: 1 Nonpersistent 2 Persistent
imqOverrideJMSExpiration	Boolean	false	Allow client-set expiration time to be overridden?
imqJMSExpiration	Long integer	0	Overriding value of expiration time, in milliseconds A value of 0 denotes an unlimited expiration time (message never expires).
imqOverrideJMSPriority	Boolean	false	Allow client-set priority level to be overridden?
imqJMSPriority	Integer	4 (normal)	Overriding value of priority level (0 to 9)
imqOverrideJMSHeadersToTemporaryDestinations	Boolean	false	Apply overrides to temporary destinations?

Destination Attributes

Table 16–9 lists the attributes that can be set for a destination administered object.

TABLE 16–9 Destination Attributes

Attribute	Type	Default	Description
imqDestinationName	String	Untitled_Destination_Object	Name of physical destination The destination name may contain only alphanumeric characters (no spaces) and must begin with an alphabetic character or the underscore (_) or dollar sign (\$) character. It may not begin with the characters mq.
imqDestinationDescription	String	None	Descriptive string for destination

SOAP Endpoint Attributes

Table 16–10 lists the attributes used to configure endpoint URLs for applications that use the Simple Object Access Protocol (SOAP); see the *Message Queue Developer's Guide for Java Clients* for more information.

TABLE 16–10 SOAP Endpoint Attributes

Attribute	Type	Default	Description
imqSOAPEndpointList	String	None	List of one or more URLs representing SOAP endpoints to which to send messages, separated by spaces Each URL should be associated with a servlet that can receive and process SOAP messages. Example: http://www.serv1/ http://www.serv2/ If the list specifies more than one URL, messages are broadcast to all of them.
imqEndpointName	String	Untitled_Endpoint_Object	Name of SOAP endpoint
imqEndpointDescription	String	None	Descriptive string for SOAP endpoint Example: My endpoints for broadcast

JMS Resource Adapter Property Reference

This chapter describes the configuration properties of the Message Queue™ JMS Resource Adapter (JMS RA), which 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 plugged into an application server, the Resource Adapter allows applications deployed in that application server to use Message Queue to send and receive JMS messages.

The Message Queue JMS Resource Adapter exposes its configuration properties through three JavaBean components:

- The ResourceAdapter JavaBean (“[ResourceAdapter JavaBean](#)” on page 303) affects the behavior of the Resource Adapter as a whole.
- The ManagedConnectionFactory JavaBean (“[ManagedConnectionFactory JavaBean](#)” on page 305) affects connections created by the Resource Adapter for use by message-driven beans (MDBs).
- The ActivationSpec JavaBean (“[ActivationSpec JavaBean](#)” on page 306) affects message endpoints that represent MDBs in their interactions with the messaging system.

To set property values for these entities, you use the tools provided by your application server for configuration and deployment of the Resource Adapter and for deployment of MDBs.

This chapter lists and describes the configuration properties of the Message Queue JMS Resource Adapter. It contains the following sections:

ResourceAdapter JavaBean

The ResourceAdapter configuration configures the default JMS Resource Adapter behavior. [Table 17–1](#) lists and describes the properties with which you can configure this JavaBean.

TABLE 17-1 Resource Adapter Properties

Property	Type	Default	Description
addressList	String	mq://localhost:7676/jms	<i>(Required)</i> Message service address for connecting to Message Queue service Equivalent to connectionURL (below); you must set one or the other.
connectionURL	String	mq://localhost:7676/jms	Message service address for connecting to the Message Queue service Equivalent to addressList (above); you must set one or the other.
userName	String	guest	<i>(Required)</i> Default user name for connecting to Message Queue service
password	String	guest	<i>(Required)</i> Default password for connecting to Message Queue service
addressListBehavior	String	PRIORITY	Order in which to attempt connection to Message Queue service: PRIORITY: Order specified in address list RANDOM: Random order Note – Reconnection attempts after a connection failure start with the broker whose connection failed and proceed sequentially through the address list, regardless of the value set for this property.
addressListIterations	Integer	1	Number of times to iterate through address list attempting to establish or reestablish a connection
reconnectEnabled	Boolean	false	Attempt to reestablish a lost connection?
reconnectAttempts	Integer	6	Number of times to attempt reconnection to each address in address list before moving on to next
reconnectInterval	Long integer	30000	Interval, in milliseconds, between reconnection attempts

ManagedConnectionFactory JavaBean

A *managed connection factory* defines the connections that the Resource Adapter provides to a message-driven bean. Table 17–2 shows the properties of the ManagedConnectionFactory JavaBean; if set, these properties override the corresponding properties of the ResourceAdapter JavaBean.

TABLE 17–2 Managed Connection Factory Properties

Property	Type	Default	Description
addressList	String	Inherited from ResourceAdapter JavaBean (see Table 17–1)	List of message service addresses for connecting to Message Queue service
userName	String	guest	(Optional) User name for connecting to Message Queue service
password	String	guest	(Optional) Password for connecting to Message Queue service
clientID	String	None	Client identifier for connections to Message Queue service
addressListBehavior	String	PRIORITY	Order in which to attempt connection to Message Queue service: PRIORITY: Order specified in address list RANDOM: Random order Note – Reconnection attempts after a connection failure start with the broker whose connection failed and proceed sequentially through the address list, regardless of the value set for this property.
addressListIterations	Integer	1	Number of times to iterate through address list attempting to establish or reestablish a connection
reconnectEnabled	Boolean	false	Attempt to reestablish a lost connection?
reconnectAttempts	Integer	6	Number of times to attempt reconnection to each address in address list before moving on to next
reconnectInterval	Long integer	30000	Interval, in milliseconds, between reconnection attempts

ActivationSpec JavaBean

Table 17–3 shows the configurable properties of the ActivationSpec JavaBean. These properties are used by the application server when instructing the Resource Adapter to activate a message endpoint and associate it with a message-driven bean.

TABLE 17–3 Activation Specification Properties

Property	Type	Default	Description
addressList ¹	String	Inherited from ResourceAdapter JavaBean	(Optional) Message service address for connecting to Message Queue service
destination ²	String	None	(Required) Name of destination from which to consume messages The value must be that of the destinationName property for a Message Queue destination administered object.
destinationType ²	String	None	(Required) Type of destination specified by destination property: javax.jms.Queue: Queue destination javax.jms.Topic: Topic destination
messageSelector ²	String	None	(Optional) Message selector for filtering messages delivered to consumer
subscriptionName ²	String	None	Name for durable subscriptions This property must be set if subscriptionDurability is set to Durable.

¹ Property specific to Message Queue JMS Resource Adapter
² Standard Enterprise JavaBean (EJB) and J2EE Connector Architecture (CA) property

TABLE 17-3 Activation Specification Properties (Continued)

Property	Type	Default	Description
subscriptionDurability ²	String	NonDurable	<p>Durability of consumer for topic destination:</p> <p>Durable: Durable consumer</p> <p>NonDurable: Nondurable consumer</p> <p>This property is valid only if destinationType is set to javax.jms.Topic, and is optional for nondurable subscriptions and required for durable ones. If set to Durable, the clientId and subscriptionName properties must also be set.</p>
clientId ²	String	None	<p>Client ID for connections to Message Queue service</p> <p>This property must be set if subscriptionDurability is set to Durable.</p>
acknowledgeMode ²	String	Auto-acknowledge	<p>(Optional) Acknowledgment mode:</p> <p>Auto-acknowledge:</p> <p>Auto-acknowledge mode</p> <p>Dups-ok-acknowledge:</p> <p>Dups-OK-acknowledge mode</p>
customAcknowledgeMode	String	None	<p>Acknowledgment mode for MDB message consumption</p> <p>Valid values are No_acknowledge or null.</p> <p>You can use no-acknowledge mode only for a nontransacted, nondurable topic subscription; if you use this setting with a transacted subscription or a durable subscription, subscription activation will fail.</p>
endpointExceptionRedeliveryAttempts	Integer	6	<p>Number of times to redeliver a message when MDB throws an exception during message delivery</p>

² Standard Enterprise JavaBean (EJB) and J2EE Connector Architecture (CA) property

TABLE 17-3 Activation Specification Properties (Continued)

Property	Type	Default	Description
sendUndeliverableMsgsToDMQ	Boolean	true	<p>Place message in dead message queue when MDB throws a runtime exception and number of redelivery attempts exceeds the value of <code>endpointExceptionRedeliveryAttempts</code>.</p> <p>If false, the Message Queue broker will attempt redelivery of the message to any valid consumer, including the same MDB.</p>

Metrics Reference

This chapter describes the metric information that a Message Queue™ message broker can provide for monitoring, tuning, and diagnostic purposes. This information can be made available in a variety of ways:

- In a log file (see “[Sending Metrics Data to Log Files](#)” on page 184)
- Via the Command utility’s `metrics bkr` command (see “[Broker Management](#)” on page 250)
- In metrics messages sent to a metrics topic destination (see “[Writing an Application to Monitor Brokers](#)” on page 190)

The tables in this chapter list the kinds of metric information available and the forms in which it can be provided. For metrics provided through the Command utility’s `metrics bkr` command, the tables list the metric type with which they can be requested; for those provided in metrics messages, the tables list the metrics topic destination to which they are delivered. The chapter consists of the following sections:

- “[JVM Metrics](#)” on page 309
- “[Brokerwide Metrics](#)” on page 310
- “[Connection Service Metrics](#)” on page 312
- “[Destination Metrics](#)” on page 313

JVM Metrics

Table 18–1 shows the metric information that the broker reports for the broker process JVM (Java Virtual Machine) heap.

TABLE 18–1 JVM Metrics

Metric Quantity	Description	Log File?	metrics bkr MetricType	Metrics Topic
JVM heap: total memory	Current total memory, in bytes	Yes	cxn	mq.metrics.jvm

TABLE 18–1 JVM Metrics (Continued)

Metric Quantity	Description	Log File?	metrics bkr MetricType	Metrics Topic
JVM heap: free memory	Amount of memory currently available for use, in bytes	Yes	cxn	mq.metrics.jvm
JVM heap: max memory	Maximum allowable heap size, in bytes	Yes	None	mq.metrics.jvm

Brokerwide Metrics

Table 18–2 shows the brokerwide metric information that the broker reports.

TABLE 18–2 Brokerwide Metrics

Metric Quantity	Description	Log File?	metrics bkr MetricType	Metrics Topic
Connections				
Num connections	Total current number of connections for all connection services	Yes	cxn	mq.metrics.broker
Num threads	Total current number of threads for all connection services	Yes	cxn	None
Min threads	Total minimum number of threads for all connection services	Yes	cxn	None
Max threads	Total maximum number of threads for all connection services	Yes	cxn	None
Stored Messages				
Num messages	Current number of payload messages stored in memory and persistent store	No	None ¹	mq.metrics.broker
Total message bytes	Total size in bytes of payload messages currently stored in memory and persistent store	No	None ¹	mq.metrics.broker
Message Flow				
Num messages in	Cumulative number of payload messages received since broker started	Yes	ttl	mq.metrics.broker
Num messages out	Cumulative number of payload messages sent since broker started	Yes	ttl	mq.metrics.broker
Rate messages in	Current rate of flow of payload messages into broker	Yes	rts	None

¹ Use query bkr command instead

TABLE 18–2 Brokerwide Metrics (Continued)

Metric Quantity	Description	Log File?	metrics bkr MetricType	Metrics Topic
Rate messages out	Current rate of flow of payload messages out of broker	Yes	rts	None
Message bytes in	Cumulative size in bytes of payload messages received since broker started	Yes	ttl	mq.metrics.broker
Message bytes out	Cumulative size in bytes of payload messages sent since broker started	Yes	ttl	mq.metrics.broker
Rate message bytes in	Current rate of flow of payload message bytes into broker	Yes	rts	None
Rate message bytes out	Current rate of flow of payload message bytes out of broker	Yes	rts	None
Num packets in	Cumulative number of payload and control packets received since broker started	Yes	ttl	mq.metrics.broker
Num packets out	Cumulative number of payload and control packets sent since broker started	Yes	ttl	mq.metrics.broker
Rate packets in	Current rate of flow of payload and control packets into broker	Yes	rts	None
Rate packets out	Current rate of flow of payload and control packets out of broker	Yes	rts	None
Packet bytes in	Cumulative size in bytes of payload and control packets received since broker started	Yes	ttl	mq.metrics.broker
Packet bytes out	Cumulative size in bytes of payload and control packets sent since broker started	Yes	ttl	mq.metrics.broker
Rate packet bytes in	Current rate of flow of payload and control packet bytes into broker	Yes	rts	None
Rate packet bytes out	Current rate of flow of payload and control packet bytes out of broker	Yes	rts	None
Destinations				
Num destinations	Current number of physical destinations	No	None	mq.metrics.broker

Connection Service Metrics

Table 18–3 shows the metric information that the broker reports for individual connection services.

TABLE 18–3 Connection Service Metrics

Metric Quantity	Description	Log File?	metrics svc MetricType	Metrics Topic
Connections				
Num connections	Current number of connections	No	cxn ¹	None
Num threads	Current number of threads	No	cxn ¹	None
Min threads	Minimum number of threads assigned to service	No	cxn	None
Max threads	Maximum number of threads assigned to service	No	cxn	None
Message Flow				
Num messages in	Cumulative number of payload messages received through connection service since broker started	No	ttl	None
Num messages out	Cumulative number of payload messages sent through connection service since broker started	No	ttl	None
Rate messages in	Current rate of flow of payload messages into broker through connection service	No	rts	None
Rate messages out	Current rate of flow of payload messages out of broker through connection service	No	rts	None
Message bytes in	Cumulative size in bytes of payload messages received through connection service since broker started	No	ttl	None
Message bytes out	Cumulative size in bytes of payload messages sent through connection service since broker started	No	ttl	None
Rate message bytes in	Current rate of flow of payload message bytes into broker through connection service	No	rts	None
Rate message bytes out	Current rate of flow of payload message bytes out of broker through connection service	No	rts	None
Num packets in	Cumulative number of payload and control packets received through connection service since broker started	No	ttl	None
Num packets out	Cumulative number of payload and control packets sent through connection service since broker started	No	ttl	None

¹ Also available with query svc command

TABLE 18–3 Connection Service Metrics (Continued)

Metric Quantity	Description	Log File?	metrics svc MetricType	Metrics Topic
Rate packets in	Current rate of flow of payload and control packets into broker through connection service	No	rts	None
Rate packets out	Current rate of flow of payload and control packets out of broker through connection service	No	rts	None
Packet bytes in	Cumulative size in bytes of payload and control packets received through connection service since broker started	No	ttl	None
Packet bytes out	Cumulative size in bytes of payload and control packets sent through connection service since broker started	No	ttl	None
Rate packet bytes in	Current rate of flow of payload and control packet bytes into broker through connection service	No	rts	None
Rate packet bytes out	Current rate of flow of payload and control packet bytes out of broker through connection service	No	rts	None

Destination Metrics

Table 18–4 shows the metric information that the broker reports for individual destinations.

TABLE 18–4 Destination Metrics

Metric Quantity	Description	Log File?	metrics dst MetricType	Metrics Topic
Message Consumers				
Num consumers	<p>Current number of associated message consumers</p> <p>For queue destinations, this attribute includes both active and backup consumers. For topic destinations, it includes both nondurable and (active and inactive) durable subscribers and is equivalent to “Num active consumers.”</p>	No	con	mq.metrics.destination.queue.queueName mq.metrics.destination.topic.topicName

TABLE 18–4 Destination Metrics (Continued)

Metric Quantity	Description	Log File?	metrics dst MetricType	Metrics Topic
Peak num consumers	<p>Peak number of associated message consumers since broker started</p> <p>For queue destinations, this attribute includes both active and backup consumers. For topic destinations, it includes both nondurable and (active and inactive) durable subscribers and is equivalent to “Peak num active consumers.”</p>	No	con	mq.metrics.destination.queue.queueName mq.metrics.destination.topic.topicName
Avg num consumers	<p>Average number of associated message consumers since broker started</p> <p>For queue destinations, this attribute includes both active and backup consumers. For topic destinations, it includes both nondurable and (active and inactive) durable subscribers and is equivalent to “Avg num active consumers.”</p>	No	con	mq.metrics.destination.queue.queueName mq.metrics.destination.topic.topicName
Num active consumers	<p>Current number of associated active message consumers</p> <p>For topic destinations, this attribute includes both nondurable and (active and inactive) durable subscribers and is equivalent to “Num consumers.”</p>	No	con	mq.metrics.destination.queue.queueName mq.metrics.destination.topic.topicName

TABLE 18–4 Destination Metrics (Continued)

Metric Quantity	Description	Log File?	metrics dst MetricType	Metrics Topic
Peak num active consumers	Peak number of associated active message consumers since broker started For topic destinations, this attribute includes both nondurable and (active and inactive) durable subscribers and is equivalent to “Peak num consumers.”	No	con	mq.metrics.destination.queue.queueName mq.metrics.destination.topic.topicName
Avg num active consumers	Average number of associated active message consumers since broker started For topic destinations, this attribute includes both nondurable and (active and inactive) durable subscribers and is equivalent to “Avg num consumers.”	No	con	mq.metrics.destination.queue.queueName mq.metrics.destination.topic.topicName
Num backup consumers ¹	Current number of associated backup message consumers	No	con	mq.metrics.destination.queue.queueName mq.metrics.destination.topic.topicName
Peak num backup consumers ¹	Peak number of associated backup message consumers since broker started	No	con	mq.metrics.destination.queue.queueName mq.metrics.destination.topic.topicName
Avg num backup consumers ¹	Average number of associated backup message consumers since broker started	No	con	mq.metrics.destination.queue.queueName mq.metrics.destination.topic.topicName

Stored Messages

Num messages	Current number of messages stored in memory and persistent store	No	con ttl rts ²	mq.metrics.destination.queue.queueName mq.metrics.destination.topic.topicName
--------------	--	----	--------------------------------	--

¹ Queue destinations only² Also available with query dst command

TABLE 18–4 Destination Metrics (Continued)

Metric Quantity	Description	Log File?	metrics dst MetricType	Metrics Topic
Peak num messages	Peak number of messages stored in memory and persistent store since broker started	No	con ttl rts	mq.metrics.destination.queue.queueName mq.metrics.destination.topic.topicName
Avg num messages	Average number of messages stored in memory and persistent store since broker started	No	con ttl rts	mq.metrics.destination.queue.queueName mq.metrics.destination.topic.topicName
Total message bytes	Current total size in bytes of messages stored in memory and persistent store	No	ttl rts ²	mq.metrics.destination.queue.queueName mq.metrics.destination.topic.topicName
Peak total message bytes	Peak total size in bytes of messages stored in memory and persistent store since broker started	No	ttl rts	mq.metrics.destination.queue.queueName mq.metrics.destination.topic.topicName
Avg total message bytes	Average total size in bytes of messages stored in memory and persistent store since broker started	No	ttl rts	mq.metrics.destination.queue.queueName mq.metrics.destination.topic.topicName
Message Flow				
Num messages in	Cumulative number of messages received since broker started	No	ttl	mq.metrics.destination.queue.queueName mq.metrics.destination.topic.topicName
Num messages out	Cumulative number of messages sent since broker started	No	ttl	mq.metrics.destination.queue.queueName mq.metrics.destination.topic.topicName
Msg bytes in	Cumulative size in bytes of messages received since broker started	No	ttl	mq.metrics.destination.queue.queueName mq.metrics.destination.topic.topicName
Msg bytes out	Cumulative size in bytes of messages sent since broker started	No	ttl	mq.metrics.destination.queue.queueName mq.metrics.destination.topic.topicName
Peak message bytes	Size in bytes of largest single message received since broker started	No	ttl rts	mq.metrics.destination.queue.queueName mq.metrics.destination.topic.topicName

² Also available with query dst command

TABLE 18–4 Destination Metrics (Continued)

Metric Quantity	Description	Log File?	metrics dst MetricType	Metrics Topic
Rate num messages in	Current rate of flow of messages received	No	rts	None
Rate num messages out	Current rate of flow of messages sent	No	rts	None
Rate msg bytes in	Current rate of flow of message bytes received	No	rts	None
Rate msg bytes out	Current rate of flow of message bytes sent	No	rts	None
Disk Utilization				
Disk reserved ³	Amount of disk space, in bytes, reserved for destination	No	dsk	mq.metrics.destination.queue.queueName mq.metrics.destination.topic.topicName
Disk used ³	Amount of disk space, in bytes, currently in use by destination	No	dsk	mq.metrics.destination.queue.queueName mq.metrics.destination.topic.topicName
Disk utilization ratio ³	Ratio of disk space in use to disk space reserved for destination	No	dsk	mq.metrics.destination.queue.queueName mq.metrics.destination.topic.topicName

³ File-based persistence only



PART IV

Appendixes

- [Appendix A](#)
- [Appendix B](#)
- [Appendix C](#)
- [Appendix D](#)

Platform-Specific Locations of Message Queue™ Data

Sun Java System™ Message Queue data is stored in different locations on different operating system platforms. The tables that follow show the location of various types of Message Queue data on the following platforms:

- “Solaris” on page 321
- “Linux” on page 322
- “Windows” on page 323

In the tables, *instanceName* denotes the name of the broker instance with which the data is associated.

Solaris

Table A-1 shows the location of Message Queue data on the Solaris operating system. If you are using Message Queue on Solaris with the standalone version of Sun Java System Application Server, the directory structure is like that described under “Windows” on page 323 .

TABLE A-1 Message Queue Data Locations on Solaris Platform

Data Category	Location
Broker instance configuration properties	/var/imq/instances/ <i>instanceName</i> /props/config.properties
Broker configuration file templates	/usr/share/lib/imq/props/broker/
Persistent store (messages, destinations, durable subscriptions, transactions)	/var/imq/instances/ <i>instanceName</i> /fs350 or a JDBC-accessible data store
Broker instance log file directory (default location)	/var/imq/instances/ <i>instanceName</i> /log/
Administered objects (object store)	Local directory of your choice or an LDAP server

TABLE A-1 Message Queue Data Locations on Solaris Platform (Continued)

Data Category	Location
Security: user repository	/var/imq/instances/ <i>instanceName</i> /etc/passwd or an LDAP server
Security: access control file (default location)	/var/imq/instances/ <i>instanceName</i> /etc/accesscontrol.properties
Security: password file directory (default location)	/var/imq/instances/ <i>instanceName</i> /etc/
Security: example password file	/etc/imq/passfile.sample
Security: broker's key store file location	/etc/imq/
JavaDoc API documentation	/usr/share/javadoc/imq/index.html
Example applications and configurations	/usr/demo/imq/
Java archive (.jar), Web archive (.war), and Resource Adapter archive (.rar) files	/usr/share/lib/

Linux

Table A-2 shows the location of Message Queue data on the Linux operating system.

TABLE A-2 Message Queue Data Locations on Linux Platform

Data Category	Location
Broker instance configuration properties	/var/opt/sun/mq/instances/ <i>instanceName</i> /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/ <i>instanceName</i> /fs350/ or a JDBC-accessible data store
Broker instance log file directory (default location)	/var/opt/sun/mq/instances/ <i>instanceName</i> /log/
Administered objects (object store)	Local directory of your choice or an LDAP server
Security: user repository	/var/opt/sun/mq/instances/ <i>instanceName</i> /etc/passwd or an LDAP server
Security: access control file (default location)	/var/opt/sun/mq/instances/ <i>instanceName</i> /etc/accesscontrol.properties
Security: password file directory (default location)	/var/opt/sun/mq/instances/ <i>instanceName</i> /etc/

TABLE A-2 Message Queue Data Locations on Linux Platform (Continued)

Data Category	Location
Security: example password file	/etc/opt/sun/mq/passfile.sample
Security: broker's key store file location	/etc/opt/sun/mq/
JavaDoc API documentation	/opt/sun/mq/javadoc/index.html
Example applications and configurations	/opt/sun/mq/examples/
Java archive (.jar), Web archive (.war), and Resource Adapter archive (.rar) files	/opt/sun/mq/share/lib/
Shared library (.so) files	/opt/sun/mq/lib/

Windows

Table A-3 shows the location of Message Queue data on the Windows operating system. The table also applies to the Solaris platform when Message Queue is bundled with the standalone version of Sun Java System Application Server. That version of Application Server is bundled with neither Solaris nor Sun Java Enterprise System. Use the path names in Table A-3, but change the direction of the slash characters from the Windows backslash (\) to the Solaris forward slash (/). See “Directory Variable Conventions” on page 25 for definitions of the IMQ_HOME and IMQ_VARHOME directory variables.

TABLE A-3 Message Queue Data Locations on Windows Platform

Data Category	Location
Broker instance configuration properties	IMQ_VARHOME\instances\instanceName\props\config.properties
Broker configuration file templates	IMQ_HOME\lib\props\broker\
Persistent store (messages, destinations, durable subscriptions, transactions)	IMQ_VARHOME\instances\instanceName\fs350\ or a JDBC-accessible data store
Broker instance log file directory (default location)	IMQ_VARHOME\instances\instanceName\log\
Administered objects (object store)	Local directory of your choice or an LDAP server
Security: user repository	IMQ_VARHOME\instances\instanceName\etc\passwd or an LDAP server
Security: access control file (default location)	IMQ_VARHOME\instances\instanceName\etc\accesscontrol.properties

TABLE A-3 Message Queue Data Locations on Windows Platform (Continued)

Data Category	Location
Security: password file directory (default location)	IMQ_HOME\etc\
Security: example password file	IMQ_HOME\etc\passfile.sample
Security: broker's key store file location	IMQ_HOME\etc\
JavaDoc API documentation	IMQ_HOME\javadoc\index.html
Example applications and configurations	IMQ_HOME\demo\
Java archive (.jar), Web archive (.war), and Resource Adapter archive (.rar) files	IMQ_HOME\lib\

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.

Classification Scheme

[Appendix B](#) 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 (for example, 3.0 or 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.

Interface Stability

Appendix B 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

TABLE B-2 Stability of Message Queue Interfaces (Continued)

Interface	Classification
Administered Object API (com.sun.messaging)	Evolving
.jar and .war Files	
imq.jar location and name	Stable
jms.jar location and name	Evolving
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
saa-api.jar location and name	Evolving
saa-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.dmqs destination	Stable
mq.metrics.* destinations	Evolving
Configuration Properties	
Message Queue JMS Resource Adapter configuration properties	Evolving

TABLE B-2 Stability of Message Queue Interfaces (Continued)

Interface	Classification
Message Queue JMS Resource Adapter JavaBean and ActivationSpec configuration properties	Evolving
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

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” on page 329](#)
- [“Enabling HTTP Support” on page 331](#)
- [“Enabling HTTPS Support” on page 337](#)
- [“Troubleshooting” on page 348](#)

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](#) shows the main components involved in providing HTTP/HTTPS support.

- On the client side, an HTTP or HTTPS transport driver encapsulates the Message Queue message into an HTTP request and makes sure that these requests are sent to the Web server/application server in the correct sequence.
- The client can use an HTTP proxy server to communicate with the broker if necessary. The proxy’s address is specified using command line options when starting the client. See [“Using an HTTP Proxy” on page 337](#) for more information.
- An HTTP or HTTPS tunnel servlet (both bundled with Message Queue) is loaded in a Web server/application server and used to pull payload messages out of client HTTP requests before forwarding them to the broker. The HTTP/HTTPS tunnel servlet also sends broker

messages back to the client in response to HTTP requests made by the client. A single HTTP/HTTPS tunnel servlet can be used to access multiple brokers.

- On the broker side, the `httpjms` or `httpsjms` connection service unwraps and demultiplexes incoming messages from the corresponding tunnel servlet.
- If the Web server/application server fails and is restarted, all connections are restored and there is no effect on clients. If the broker fails and is restarted, an exception is thrown and clients must re-establish their connections. In the unlikely case that both the Web server/application server and the broker fail, and the broker is not restarted, the Web server/application server will restore client connections and continue waiting for a broker connection—without notifying clients. To avoid this situation, always restart the broker.

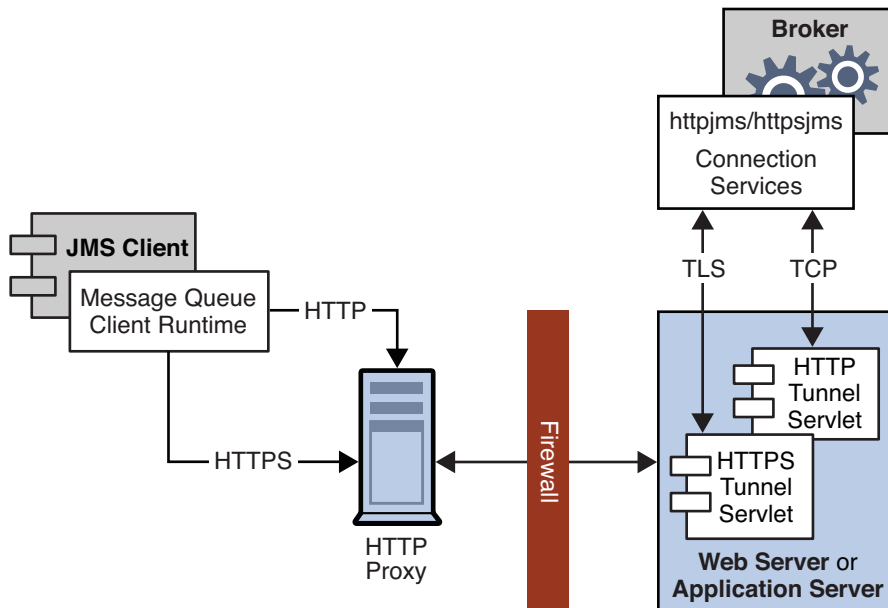


FIGURE C-1 HTTP/HTTPS Support Architecture

As you can see from [Figure C-1](#), the architecture for HTTP and HTTPS support is very similar. The main difference is that, in the case of HTTPS (`httpsjms` connection service), the tunnel servlet has a secure connection to both the client application and broker.

The secure connection to the broker is provided through an SSL-enabled tunnel servlet—Message Queue’s **HTTPS tunnel servlet**—which passes a self-signed certificate to any broker requesting a connection. The certificate is used by the broker to set up an encrypted connection to the **HTTPS tunnel servlet**. Once this connection is established, a secure connection between a client application and the tunnel servlet can be negotiated by the client application and the Web server/application server.

Enabling HTTP Support

The following sections describe the steps you need to take to enable HTTP support.

▼ To Enable HTTP Support

- 1 **Deploy the HTTP tunnel servlet.** You can deploy the HTTP tunnel servlet on the following:
 - Sun Java System™ Web Server
 - Sun Java System Application Server
- 2 **Configure the broker's httpjms connection service and start the broker.**
- 3 **Configure an HTTP connection.**

Step 1. Deploy the HTTP Tunnel Servlet

You can deploy the HTTP tunnel servlet as a Web archive (.war) file on a Sun Java System Web Server or Sun Java System Application Server.

Deploying the HTTP tunnel servlet as a .war file consists of using the deployment mechanism provided by the Web server/application server. The HTTP tunnel servlet .war file (imhttp.war) is located in the directory containing .jar, .war, and .rar files, and depends on your operating system (see [Appendix A](#)).

The .war file includes a deployment descriptor that contains the basic configuration information needed by the Web server/application server to load and run the servlet. Depending on the Web server/application server, you might also need to specify the context root portion of the servlet's URL.

Deploying as a Web Archive File

For deployment on a Sun Java System Web Server, see “[Deploying the HTTP Tunnel Servlet on Sun Java System Web Server](#)” on page 331.

For deployment on a Sun Java System Application Server, see “[Deploying the HTTP Tunnel Servlet on Sun Java System Application Server](#)” on page 333.

Deploying the HTTP Tunnel Servlet on Sun Java System Web Server

The instructions below refer to deployment on Sun Java System Web Server. You can verify successful HTTP tunnel servlet deployment by accessing the servlet URL using a Web browser. It should display status information.

▼ To Deploy the HTTP Tunnel Servlet as a .war File

- 1 In the browser-based administration GUI, select the Virtual Server Class tab and select Manage Classes.
- 2 Select the appropriate virtual server class name (for example, `defaultClass`) and click the Manage button.
- 3 Select Manage Virtual Servers.
- 4 Select an appropriate virtual server name and click the Manage button.
- 5 Select the Web Applications tab.
- 6 Click on Deploy Web Application.
- 7 Select the appropriate values for the WAR File On and WAR File Path fields so as to point to the `imqhttp.war` file, which can be found in a directory that depends on your operating system (see [Appendix A](#)).

- 8 Enter a path in the Application URI field.

The Application URI field value is the `/contextRoot` portion of the tunnel servlet URL:

```
http://hostName:portNumber/contextRoot/tunnel
```

For example, if you set the `contextRoot` to `imq`, the Application URI field would be:

```
/imq
```

- 9 Enter the installation directory path (typically somewhere under the Sun Java System Web Server installation root) where the servlet should be deployed.
- 10 Click OK.
- 11 Restart the Web server instance.

The servlet is now available at the following address:

```
http://hostName:portNumber/  
contextRoot/tunnel
```

Clients can now use this URL to connect to the message service using an HTTP connection.

Disabling a Server Access Log

You do not have to disable the server access log, but you will obtain better performance if you do.

▼ To Disable the Server Access Log

- 1 Select the Status tab.
- 2 Choose the Log Preferences Page.
Use the Log client accesses control to disable logging.

Deploying the HTTP Tunnel Servlet on Sun Java System Application Server

This section describes how you deploy the HTTP tunnel servlet as a `.war` file on the Sun Java System Application Server, and then configure the tunnel servlet to accept connections from a Message Queue broker.

Two steps are required:

- Deploy the HTTP tunnel servlet using the Application Server deployment tool.
- Modify the application server instance's `server.policy` file.

Using the Deployment Tool

▼ To Deploy the HTTP Tunnel Servlet in an Application Server Environment

- 1 In the Web-based administration GUI, choose
App Server > Instances > `server1` > Applications > Web Applications.
- 2 Click the Deploy button.
- 3 In the File Path: text field, enter the location of the HTTP tunnel servlet `.war` file (`imqhttp.war`), and click OK.

The location of the `imqhttp.war` file depends on your operating system (see [Appendix A](#)).

- 4 Set the value for the Context Root text field, and click OK.

The Context Root field value is the `/contextRoot` portion of the tunnel servlet URL:

`http://hostName:portNumber/contextRoot/tunnel`

For example, you could set the Context Root field to `/imq`.

The confirmation screen that appears confirms that the tunnel servlet has been successfully deployed, is enabled by default, and in this case is located at:

```
/var/opt/SUNWappserver8/domains/domain1/server1/applications/
j2ee-modules/imqhttp_1
```

The servlet is now available at the following URL:

```
http://hostName:portNumber/  
contextRoot/tunnel
```

Clients can now use this URL to connect to the message service using an HTTP connection.

Modifying the server.policy File

The Application Server enforces a set of default security policies that, unless modified, would prevent the HTTP tunnel servlet from accepting connections from the Message Queue broker.

Each application server instance has a file that contains its security policies, or rules. For example, the location of this file for the server1 instance on Solaris is:

```
/var/opt/SUNWappserver8/domains/domain1/server1/config/  
server.policy
```

To configure the tunnel servlet to accept connections from the Message Queue broker, an additional entry is required in this file.

▼ To Modify the Application Server's server.policy File

- 1 Open the `server.policy` file.

- 2 Add the following entry:

```
grant codeBase  
"file:/var/opt/SUNWappserver8/domains/domain1/server1/  
applications/j2ee-modules/imqhttp_1/-"  
{  
    permission java.net.SocketPermission "*",  
        "connect,accept,resolve";  
};
```

Step 2. Configure the httpjms Connection Service

HTTP support is not activated for a broker by default, so you need to reconfigure the broker to activate the httpjms connection service. Once reconfigured, the broker can be started as outlined in [“Starting Brokers” on page 68](#).

▼ 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](#)):

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

2 Add the `httpjms` value to the `imq.service.activelist` property:

```
imq.service.activelist=jms,admin,httpjms
```

At startup, the broker looks for a Web server/application server and HTTP tunnel servlet running on its host machine. To access a remote tunnel servlet, however, you can reconfigure the `servletHost` and `servletPort` connection service properties.

You can also reconfigure the `pullPeriod` property to improve performance. The `httpjms` connection service configuration properties are detailed in [“Step 2. Configure the `httpjms` Connection Service” on page 334](#).

Property	Description
<code>imq.httpjms.http.servletHost</code>	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: <code>localhost</code> .
<code>imq.httpjms.http.servletPort</code>	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.
<code>imq.httpjms.http.pullPeriod</code>	Specifies the interval, in seconds, between HTTP requests made by a client runtime to pull messages from the broker. (Note that this property is set on the broker and propagates to the client runtime.) If the value is zero or negative, the client keeps one HTTP request pending at all times, ready to pull messages as fast as possible. With a large number of clients, this can be a heavy drain on Web/application server resources and the server may become unresponsive. In such cases, you should set the <code>pullPeriod</code> property to a positive number of seconds. This sets the time the client's HTTP transport driver waits before making subsequent pull requests. Setting the value to a positive number conserves Web/application server resources at the expense of the response times observed by clients. Default: -1.

Property	Description
<code>imq.httpjms.http.connectionTimeout</code>	Specifies the time, in seconds, that the client runtime waits for a response from the HTTP tunnel servlet before throwing an exception. (Note that this property is set on the broker and propagates to the client runtime.) This property also specifies the time the broker waits after communicating with the HTTP tunnel servlet before freeing up a connection. A timeout is necessary in this case because the broker and the tunnel servlet have no way of knowing if a client that is accessing the HTTP servlet has terminated abnormally. Default: 60.

Step 3. Configure an HTTP Connection

A client application must use an appropriately configured connection factory administered object to make an HTTP connection to a broker. This section discusses HTTP connection configuration issues.

Configuring the Connection Factory

To enable HTTP support, you need to set the connection factory's `imqAddressList` attribute to the HTTP tunnel servlet URL. The general syntax of the HTTP tunnel servlet URL is the following:

```
http://hostName:portNumber  
  
/contextRoot/tunnel
```

where *hostName:portNumber* is the name and port of the Web server/application server hosting the HTTP tunnel servlet and *contextRoot* is a path set when deploying the tunnel servlet on the Web server/application server.

For more information on connection factory attributes in general, and the `imqAddressList` attribute in particular, see the *Message Queue Developer's Guide for Java Clients*.

You can set connection factory attributes in one of the following ways:

- Using the `-o` option to the `imqobjmgr` command that creates the connection factory administered object (see [“Adding a Connection Factory” on page 164](#)), or set the attribute when creating the connection factory administered object using the Administration Console (`imqadmin`).
- Using the `-D` option to the command that launches the client (see the *Message Queue Developer's Guide for Java Clients*).
- Using an API call to set the attributes of a connection factory after you create it programmatically in client code (see the *Message Queue Developer's Guide for Java Clients*).

Using a Single Servlet to Access Multiple Brokers

You do not need to configure multiple Web servers/application servers and servlet instances if you are running multiple brokers. You can share a single Web server/application server and HTTP tunnel servlet instance among concurrently running brokers. If multiple broker instances are sharing a single tunnel servlet, you must configure the `imqAddressList` connection factory attribute as shown below:

```
http://hostName:portNumber

/contextRoot/tunnel?ServerName=
bkrHostName:instanceName
```

Where *bkrHostName* is the broker instance host name and *instanceName* is the name of the specific broker instance you want your client to access.

To check that you have entered the correct strings for *bkrHostName* and *instanceName*, generate a status report for the HTTP tunnel servlet by accessing the servlet URL from a browser. The report lists all brokers being accessed by the servlet:

```
HTTP tunnel servlet ready.
Servlet Start Time : Thu May 30 01:08:18 PDT 2005
Accepting TCP connections from brokers on port : 7675
Total available brokers = 2
Broker List :
    jpgserv:broker2
    cochin:broker1
```

Using an HTTP Proxy

If you are using an HTTP proxy to access the HTTP tunnel servlet:

- Set `http.proxyHost` system property to the proxy server host name.
- Set `http.proxyPort` system property to the proxy server port number.

You can set these properties using the `-D` option to the command that launches the client application.

Enabling HTTPS Support

The following sections describe the steps to enable HTTPS support. They are similar to those in [“Enabling HTTP Support” on page 331](#) with the addition of steps needed to generate and access SSL certificates.

▼ To Enable HTTPS Support

- 1 **Generate a self-signed certificate for the HTTPS tunnel servlet.**
- 2 **Modify the HTTP tunnel servlet .war file's deployment descriptor to:**
 - point to the location where you have placed the certificate key store
 - specify the certificate key store password
- 3 **Deploy the HTTP tunnel servlet. You can deploy the HTTP tunnel servlet on the following:**
 - Sun Java System Web Server
 - Sun Java System Application Server
- 4 **Configure the broker's httpsjms connection service and start the broker.**
- 5 **Configure an HTTPS connection.**

Each of these steps is discussed in more detail in the sections that follow.

Step 1. Generating a Self-signed Certificate for the HTTPS Tunnel Servlet

Message Queue's SSL support is oriented toward securing on-the-wire data with the assumption that the client is communicating with a known and trusted server. Therefore, SSL is implemented using only self-signed server certificates. In the httpsjms connection service architecture, the HTTPS tunnel servlet plays the role of server to both broker and application client.

Run the `keytool` utility to generate a self-signed certificate for the tunnel servlet. Enter the following at the command prompt:

```
JRE_HOME/bin/keytool -servlet keyStoreLocation
```

The utility will prompt you for the information it needs. (On Unix systems you may need to run `keytool` as the superuser (root) in order to have permission to create the key store.)

First, `keytool` prompts you for a key store password, and then it prompts you for some organizational information, and then it prompts you for confirmation. After it receives the confirmation, it pauses while it generates a key pair. It then asks you for a password to lock the particular key pair (key password); you should enter Return in response to this prompt: this makes the key password the same as the key store password.

Note – Remember the password you provide: you must provide this password later to the tunnel servlet so it can open the key store.

The JDK `keytool` utility generates a self-signed certificate and places it in Message Queue's key store file located as specified in the *keyStoreLocation* argument.

Note – The HTTPS tunnel servlet must be able to see the key store. Make sure you move/copy the generated key store located in *keyStoreLocation* to a location accessible by the HTTPS tunnel servlet (see [“Step 3. Deploying the HTTPS Tunnel Servlet” on page 340](#)).

Step 2. Modifying the HTTP Tunnel Servlet .war File's Descriptor File

The HTTP Tunnel Servlet's .war file includes a deployment descriptor that contains the basic configuration information needed by the Web server/application server to load and run the servlet.

The deployment descriptor of the `imqhttps.war` file cannot know where you have placed the key store file needed by the tunnel servlet. This requires you to edit the tunnel servlet's deployment descriptor (an XML file) to specify the key store location and password before deploying the `imqhttps.war` file.

▼ To Modify the HTTPS Tunnel Servlet .war File

1 Copy the .war file to a temporary directory.

```
cp /usr/share/lib/imq/imqhttps.war /tmp (Solaris)
cp /opt/sun/mq/share/lib/imqhttps.war /tmp (Linux)
cp IMQ_HOME/lib/imqhttps.war /tmp (Windows)
```

2 Make the temporary directory your current directory.

```
$ cd /tmp
```

3 Extract the contents of the .war file.

```
$ jar xvf imqhttps.war
```

4 List the .war file's deployment descriptor.

```
$ ls -l WEB-INF/web.xml
```

- 5 **Edit the `web.xml` file to provide correct values for the `keystoreLocation` and `keystorePassword` arguments (as well as `servletPort` and `servletHost` arguments, if necessary).**

- 6 **Reassemble the contents of the `.war` file.**

```
$ jar uvf imqhttps.war WEB-INF/web.xml
```

You are now ready to use the modified `imqhttps.war` file to deploy the HTTPS tunnel servlet. (If you are concerned about exposure of the key store password, you can use file system permissions to restrict access to the `imqhttps.war` file.)

Step 3. Deploying the HTTPS Tunnel Servlet

You can deploy the HTTP tunnel servlet as a Web archive (WAR) file on a Sun Java System Web Server or Sun Java System Application Server.

Deploying the HTTPS tunnel servlet as a `.war` file consists of using the deployment mechanism provided by the Web server/application server. The HTTPS tunnel servlet `.war` file (`imqhttps.war`) is located in a directory that depends on your operating system (see [Appendix A](#)).

You should make sure that encryption is activated for the Web server, enabling end-to-end secure communication between the client and broker.

Deploying as a Web Archive File

For deployment on a Sun Java System Web Server, see [“Deploying the HTTPS Tunnel Servlet on Sun Java System Web Server” on page 340](#).

For deployment on a Sun Java System Application Server, see [“Deploying the HTTPS Tunnel Servlet on Sun Java System Application Server” on page 342](#).

Deploying the HTTPS Tunnel Servlet on Sun Java System Web Server

This section describes how you deploy the HTTPS tunnel servlet as a `.war` file on the Sun Java System Web Server. You can verify successful HTTPS tunnel servlet deployment by accessing the servlet URL using a Web browser. It should display status information.

Before deploying the HTTPS tunnel servlet, make sure that JSSE `.jar` files are included in the Web server's classpath. The simplest way to do this is to copy the files `jsse.jar`, `jnet.jar`, and `jcert.jar` to `WebServer_TOPDIR/bin/https/jre/lib/ext`.

▼ To Deploy the https Tunnel Servlet as a `.war` File

- 1 **In the browser-based administration GUI, select the Virtual Server Class tab. Click Manage Classes.**

- 2 Select the appropriate virtual server class name (for example, `defaultClass`) and click the **Manage** button.
- 3 Select **Manage Virtual Servers**.
- 4 Select an appropriate virtual server name and click the **Manage** button.
- 5 Select the **Web Applications** tab.
- 6 Click on **Deploy Web Application**.
- 7 Select the appropriate values for the **WAR File On** and **WAR File Path** fields so as to point to the modified `imqhttps.war` file (see [“Step 2. Modifying the HTTP Tunnel Servlet .war File’s Descriptor File” on page 339.](#))
- 8 Enter a path in the **Application URI** field.

The Application URI field value is the `/contextRoot` portion of the tunnel servlet URL:

`https://hostName:portNumber/contextRoot/tunnel`

For example, if you set the `contextRoot` to `imq`, the Application URI field would be:

`/imq`

- 9 Enter the installation directory path (typically somewhere under the Sun Java System Web Server installation root) where the servlet should be deployed.
- 10 Click **OK**.
- 11 Restart the Web server instance.

The servlet is now available at the following URL:

`https://hostName:portNumber/imq/tunnel`

Clients can now use this URL to connect to the message service using a secure HTTPS connection.

Disabling a Server Access Log

You do not have to disable the server access log, but you will obtain better performance if you do.

▼ To Disable the Server Access Log

- 1 **Select the Status tab.**
- 2 **Choose the Log Preferences Page.**
Use the Log client accesses control to disable logging.

Deploying the HTTPS Tunnel Servlet on Sun Java System Application Server

This section describes how you deploy the HTTPS tunnel servlet as a `.war` file on the Sun Java System Application Server.

Two steps are required:

- Deploy the HTTPS tunnel servlet using the Application Server deployment tool.
- Modify the application server instance's `server.policy` file.

Using the Deployment Tool

The following procedure shows how to deploy the HTTPS tunnel servlet in an Application Server environment.

▼ To Deploy the HTTPS Tunnel Servlet in an Application Server Environment

- 1 **In the Web-based administration GUI, choose**
App Server > Instances > `server1` > Applications > Web Applications
- 2 **Click the Deploy button.**
- 3 **In the File Path: text field, enter the location of the HTTPS tunnel servlet `.war` file (`imqhttps.war`), and click OK.**
The location of the `imqhttps.war` file depends on your operating system (see [Appendix A](#)).
- 4 **Set the value for the Context Root text field, and click OK.**
The Context Root field value is the `/contextRoot` portion of the tunnel servlet URL:
`https://hostName:portNumber/contextRoot/tunnel`
For example, you could set the Context Root field to:
`/imq`

The next screen shows that the tunnel servlet has been successfully deployed, is enabled by default, and in this case is located at:

```
/var/opt/SUNWappserver8/domains/domain1/server1/applications/
j2ee-modules/imqhttps_1
```

The servlet is now available at the following URL:

```
https://hostName:portNumber/
contextRoot/tunnel
```

Clients can now use this URL to connect to the message service using an HTTPS connection.

Modifying the server.policy file

Application Server enforces a set of default security policies that unless modified would prevent the HTTPS tunnel servlet from accepting connections from the Message Queue broker.

Each application server instance has a file that contains its security policies or rules. For example, the location of this file for the server1 instance on Solaris is:

```
/var/opt/SUNWappserver8/domains/domain1/server1/config/
server.policy
```

To make the tunnel servlet accept connections from the Message Queue broker, an additional entry is required in this file.

▼ To Modify the Application Server's server.policy File

- 1 Open the server.policy file.

- 2 Add the following entry:

```
grant codeBase
"file:/var/opt/SUNWappserver8/domains/domain1/server1/
    applications/j2ee-modules/imqhttps_1/-"
{
    permission java.net.SocketPermission "*",
        "connect,accept,resolve";
};
```

Step 4. Configuring the httpsjms Connection Service

HTTPS support is not activated for a broker by default, so you need to reconfigure the broker to activate the httpsjms connection service. Once reconfigured, the broker can be started as outlined in [“Starting Brokers” on page 68](#).

▼ **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](#)):

```
.../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 “[Step 4. Configuring the httpsjms Connection Service](#)” on page 343.

Property	Description
imq.httpsjms.https.servletHost	Change this value, if necessary, to specify the name of the host (hostname or IP address) on which the HTTPS tunnel servlet is running. (This can be a remote host or a specific hostname on a local host.) Default: localhost.
imq.httpsjms.https.servletPort	Change this value to specify the port number that the broker uses to access the HTTPS tunnel servlet. (If the default port is changed on the Web server, you must change this property accordingly.) Default: 7674.
imq.httpsjms.https.pullPeriod	Specifies the interval, in seconds, between HTTP requests made by each client to pull messages from the broker. (Note that this property is set on the broker and propagates to the client runtime.) If the value is zero or negative, the client keeps one HTTP request pending at all times, ready to pull messages as fast as possible. With a large number of clients, this can be a heavy drain on Web server resources and the server may become unresponsive. In such cases, you should set the <code>pullPeriod</code> 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.

Property	Description
<code>imq.httpsjms.https.connectionTimeout</code>	Specifies the time, in seconds, that the client runtime waits for a response from the HTTPS tunnel servlet before throwing an exception. (Note that this property is set on the broker and propagates to the client runtime.) This property also specifies the time the broker waits after communicating with the HTTPS tunnel servlet before freeing up a connection. A timeout is necessary in this case because the broker and the tunnel servlet have no way of knowing if a client that is accessing the HTTPS servlet has terminated abnormally. Default: 60.

Step 5. Configuring an HTTPS Connection

A client application must use an appropriately configured connection factory administered object to make an HTTPS connection to a broker.

However, the client must also have access to SSL libraries provided by the Java Secure Socket Extension (JSSE) and must also have a root certificate. The SSL libraries are bundled with JDK 1.4. If you have an earlier JDK version, see [“Configuring JSSE” on page 345](#) otherwise proceed to [“Importing a Root Certificate” on page 346](#)

Once these issues are resolved, you can proceed to configuring the HTTPS connection.

Configuring JSSE

▼ To Configure JSSE

- Copy the JSSE .jar files to the JRE_HOME/lib/ext directory.**
`jsse.jar, jnet.jar, jcert.jar`
- 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).
- If not using JDK1.4, you need to set the following JSSE property using the -D option to the command that launches the client application:**
`java.protocol.handler.pkgs=com.sun.net.ssl.internal.www.protocol`

Importing a Root Certificate

If the root certificate of the CA who signed your Web server's certificate is not in the trust database by default or if you are using a proprietary Web server/application server certificate, you must add that certificate to the trust database. If this is the case, follow the instruction below, otherwise go to [“Configuring the Connection Factory” on page 346](#)

Assuming that the certificate is saved in *certFile* and that *trustStoreFile* is your key store, run the following command:

```
JRE_HOME/bin/keytool -import -trustcacerts  
-alias aliasForCertificate -file certFile  
  
-keystore trustStoreFile
```

Answer YES to the question: Trust this certificate?

You also need to specify the following JSSE properties using the `-D` option to the command that launches the client application:

```
javax.net.ssl.trustStore=trustStoreFile  
javax.net.ssl.trustStorePassword=trustStorePasswd
```

Configuring the Connection Factory

To enable HTTPS support, you need to set the connection factory's `imqAddressList` attribute to the HTTPS tunnel servlet URL. The general syntax of the HTTPS tunnel servlet URL is the following:

```
https://hostName:portNumber  
  
/contextRoot/tunnel
```

where *hostName:portNumber* is the name and port of the Web server hosting the HTTPS tunnel servlet and *contextRoot* is a path set when deploying the tunnel servlet on the Web server.

For more information on connection factory attributes in general, and the `imqAddressList` attribute in particular, see the *Message Queue Developer's Guide for Java Clients*.

You can set connection factory attributes in one of the following ways:

- Using the `-o` option to the `imqobjmgr` command that creates the connection factory administered object (see [“Adding a Connection Factory” on page 164](#)), or set the attribute when creating the connection factory administered object using the Administration Console (`imqadmin`).
- Using the `-D` option to the command that launches the client application (see the *Message Queue Developer's Guide for Java Clients*).

- Using an API call to set the attributes of a connection factory after you create it programmatically in client application code (see the *Message Queue Developer's Guide for Java Clients*).

Using a Single Servlet to Access Multiple Brokers

You do not need to configure multiple Web servers and servlet instances if you are running multiple brokers. You can share a single Web server and HTTPS tunnel servlet instance among concurrently running brokers. If multiple broker instances are sharing a single tunnel servlet, you must configure the `imqAddressList` connection factory attribute as shown below:

```
https://hostName:portNumber

/contextRoot/tunnel?ServerName=
bkrHostName:instanceName
```

Where *bkrHostName* is the broker instance host name and *instanceName* is the name of the specific broker instance you want your client to access.

To check that you have entered the correct strings for *bkrhostName* and *instanceName*, generate a status report for the HTTPS tunnel servlet by accessing the servlet URL from a browser. The report lists all brokers being accessed by the servlet:

```
HTTPS tunnel servlet ready.
Servlet Start Time : Thu May 30 01:08:18 PDT 2002
Accepting secured connections from brokers on port : 7674
Total available brokers = 2
Broker List :
    jpgserv:broker2
    cochin:broker1
```

Using an HTTP Proxy

If you are using an HTTP proxy to access the HTTPS tunnel servlet:

- Set `http.proxyHost` system property to the proxy server host name.
- Set `http.proxyPort` system property to the proxy server port number.

You can set these properties using the `-D` option to the command that launches the client application.

Troubleshooting

This section describes possible problems with an HTTP or HTTPS connection and provides guidance on how to handle them.

Server or Broker Failure

If the Web server fails and is restarted, all connections are restored and there is no effect on clients. However, if the broker fails and is restarted, an exception is thrown and clients must re-establish their connections.

If both the Web server and the broker fail, and the broker is not restarted, the Web server restores client connections and continues waiting for a broker connection without notifying clients. To avoid this situation, always make sure the broker is restarted.

Client Failure to Connect Through the Tunnel Servlet

If an HTTPS client cannot connect to the broker through the tunnel servlet, do the following:

▼ If a Client Cannot Connect

- 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.

Frequently Used Command Utility Commands

This appendix lists some frequently used Message Queue™ Command utility (`imqcmd`) commands. For a comprehensive list of command options and attributes available to you from the command line, refer to “[Command Utility](#)” on page 248 in “[Command Utility](#)” on page 248

Syntax

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

-H or -h provides comprehensive help. The -v subcommand provides version information.

When you use `imqcmd`, the Command utility prompts you for a password. To avoid the prompt (and to increase security), you can use the `-passfile pathToPassfile` option to point the utility to a password file that contains the administrator username and password.

Example: `imqcmd query bkr -u adminUserName -passfile pathToPassfile -b myServer:7676`

Broker and Cluster Management

```
imqcmd query bkr
imqcmd pause bkr
imqcmd restart bkr
imqcmd resume bkr
imqcmd shutdown bkr -b myBroker:7676
imqcmd update bkr -o "imq.system.max_count=1000"
imqcmd reload cls
```

Broker Configuration Properties (-o option)

“[Broker Configuration Properties \(-o option\)](#)” on page 350 lists frequently used broker configuration properties. For a full list of broker configuration properties and their descriptions, see [Chapter 14](#)

TABLE D-1 Broker Configuration Properties (-o option)

Property	Notes
imq.autocreate.queue	
imq.autocreate.queue.maxNumActiveConsumers	Specify -1 for unlimited
imq.autocreate.queue.maxNumBackupConsumers	Specify -1 for unlimited
imq.autocreate.topic	
imq.cluster.url	
imq.destination.DMQ.truncateBody	
imq.destination.logDeadMessages	
imq.log.file.rolloverbytes	Specify -1 for unlimited
imq.log.file.rolloversecs	Specify -1 for unlimited
imq.log.level	NONEERRORWARNINGINFO
imq.message.max_size	Specify -1 for unlimited
imq.portmapper.port	
imq.system.max_count	Specify -1 for unlimited
imq.system.max_size	Specify -1 for unlimited

Service and Connection Management

```
imqcmd list svc
imqcmd query svc
imqcmd update svc -n jms -o "minThreads=200" -o "maxThreads=400" -o "port=8995"
imqcmd pause svc -n jms
imqcmd resume svc -n jms
imqcmd list cxn -svn jms
imqcmd query cxn -n 1234567890
```

Durable Subscriber Management

```
imqcmd list dur -d MyTopic
imqcmd destroy dur -n myDurSub -c "clientID-111.222.333.444"
imqcmd purge dur -n myDurSub -c "clientID-111.222.333.444"
```

Transaction Management

```
imqcmd list txn
imqcmd commit txn -n 1234567890
imqcmd query txn -n 1234567890
imqcmd rollback txn -n 1234567890
```

Destination Management

```
imqcmd create dst -n MyQueue -t q -o "maxNumMsgs=1000" -o "maxNumProducers=5"
imqcmd update dst -n MyTopic -t t -o "limitBehavior=FLOW_CONTROL| REMOVE_OLDEST|REJECT_NEWEST|REMOVE_LOW_PRIORITY"
imqcmd compact dst -n MyQueue -t q
imqcmd purge dst -n MyQueue -t q
imqcmd pause dst -n MyQueue -t q -pst PRODUCERS|CONSUMERS|ALL
imqcmd resume dst -n MyQueue -t q
imqcmd destroy dst -n MyQueue -t q
imqcmd query dst -n MyQueue -t q
imqcmd list dst -tmp
```

Destination Configuration Properties (-o option)

“[Destination Configuration Properties \(-o option\)](#)” on page 351 lists frequently used destination configuration properties. For a full list of destination configuration properties and their descriptions, see [Chapter 15](#)

TABLE D-2 Destination Configuration Properties (-o option)

Property	Notes
consumerFlowLimit	Specify -1 for unlimited
isLocalOnly (create only)	
limitBehavior	FLOW_CONTROLREMOVE_OLDESTREJECT_NEWESTREMOVE_LOW_PRIORITY
localDeliveryPreferred (queue only)	

TABLE D-2 Destination Configuration Properties (-o option) (Continued)

Property	Notes
maxNumActiveConsumers (queue only)	Specify -1 for unlimited
maxNumBackupConsumers (queue only)	Specify -1 for unlimited
maxBytesPerMsg	Specify -1 for unlimited
maxNumMsgs	Specify -1 for unlimited
maxNumProducers	Specify -1 for unlimited
maxTotalMsgBytes	Specify -1 for unlimited
useDMQ	

Metrics

```
imqcmd metrics bkr -m cxn|rts|ttl -int 5 -msp 20
imqcmd metrics svc -m cxn|rts|ttl
imqcmd metrics dst -m con|dsk|rts|ttl
```


Index

A

- access control file
 - access rules, 137
 - format of, 136
 - location, 322, 323
 - use for, 135
 - version, 136
- access rules, 137
- acknowledgeMode activation specification
 - attribute, 307
- ActivationSpec JavaBean, 306
- addressList activation specification attribute, 306
- addressList managed connection factory attribute, 305
- addressList Resource Adapter attribute, 304
- addressListBehavior managed connection factory
 - attribute, 305
- addressListBehavior Resource Adapter attribute, 304
- addressListIterations managed connection factory
 - attribute, 305
- addressListIterations Resource Adapter attribute, 304
- admin connection service, 76, 103
- admin group, 130
- ADMIN service type, 76
- admin user, 128, 132, 134-135
- administered objects
 - attributes (reference), 293
 - deleting, 165
 - listing, 166
 - managing, 153
 - object stores
 - See object stores
 - querying, 166
- administered objects (*Continued*)
 - queue
 - See queues
 - required information, 163
 - topic
 - See topics
 - updating, 167
 - XA connection factory
 - See connection factory administered objects
- Administration Console
 - starting, 40
 - tutorial, 39
- administration tasks
 - development environment, 33-34
 - production environment, 34-35
- administration tools, 36-37
 - Administration Console, 37
 - command line utilities, 36
- administrator password, 132
- anonymous group, 130
- API documentation, 322, 323, 324
- attributes of physical destinations, 289-291
- audit logging, 152
- authentication
 - See also access control
 - about, 84
 - managing, 127-135
- authorization
 - See also access control
 - about, 84-85
 - managing, 135-141
 - user groups, 84

auto-create physical destinations

- access control, 85, 140-141
- properties (table), 267-269

automatic reconnection

- attributes for, 158
- limitations, 159

AUTOSTART property, 70

B

benchmarks, performance, 196-197

bottlenecks, performance, 200

broker clusters

- adding brokers to, 174
- architecture, 206-207
- configuration change record, 176
- configuration file, 171, 172, 173, 283
- configuration properties, 171, 282-283
- connecting brokers, 173
- pausing physical destinations, 119
- performance effect of, 207
- reasons for using, 207
- replication of physical destinations, 116
- secure interbroker connections, 174

broker components

- connection services, 75, 76
- monitoring services, 75, 85-87
- persistence services, 75, 79-82
- routing services, 75, 78
- security services, 75, 82-85

broker failure and secure connections, 348

broker metrics

- logger properties, 86, 184, 281
- metric quantities (table), 310-311
- metrics messages, 87
- reporting interval, logger, 247
- using broker log files, 185
- using imqcmd, 102, 188, 190
- using message-based monitoring, 190

broker monitoring service, properties, 278-282

broker responses, wait period for client, 297

brokers

- access control
- See* authorization

brokers (*Continued*)

- auto-create physical destination properties, 267-269
- automatically restarting, 70
- clock synchronization, 67
- clusters
 - See* broker clusters
- configuration files
 - See* configuration files
- connecting, 173
- dead message queue, 124
- displaying properties of, 98-99
- HTTP support, 331
- HTTPS support, 338
- instance configuration properties, 89
- instance name, 244
- interconnected
 - See* broker clusters
- limit behaviors, 78, 207
- listing connection services, 103
- logging
 - See* logger
- managing, 95
- memory management, 78, 115, 207
- message capacity, 78, 99, 265, 287
- message flow control
 - See* message flow control
- metrics
 - See* broker metrics
- pausing, 100-101, 250
- permissions required for starting, 68
- properties (reference), 263, 289
- querying, 98
- recovery from failure, 79
- removing, 72
- restarting, 79, 101, 250
- resuming, 100-101, 101, 250
- running as Windows service, 70
- shutting down, 101
- startup with SSL, 145
- updating properties of, 99-100

C

certificates, 141-146, 338

- client applications
 - example, 322, 323, 324
 - factors affecting performance, 200-204
- client identifier (ClientID), 159-160
 - in destroying durable subscription, 109
- client runtime
 - configuration of, 208
 - message flow tuning, 213
- clientID activation specification attribute, 307
- clientID managed connection factory attribute, 305
- clients
 - clock synchronization, 67
 - starting, 73
- clock synchronization, 67
- cluster configuration file, 171, 172, 173, 283
- cluster configuration properties, 171, 282
- cluster connection service, 141, 174
 - host name or IP address for, 172, 283
 - network transport for, 172, 283
 - port number for, 172, 283
- clusters, *See* broker clusters
- command files, 167
- command line syntax, 243
- command line utilities
 - about, 36
 - basic syntax, 243
 - displaying version, 256
 - help, 256
 - imqbrokerd, *See* imqbrokerd command, 36
 - imqcmd, *See* imqcmd command, 36
 - imqdbmgr *See* imqdbmgr command, 36
 - imqkeytool, *See* imqkeytool command, 36
 - imqobjmgr, *See* imqobjmgr command, 36
 - imqsvcadm, *See* imqsvcadm command, 36
 - imqusermgr, *See* imqusermgr command, 36
- command options, as configuration overrides, 73
- compacting
 - file-based data store, 81
 - physical destinations, 121-123
- config.properties file, 89, 174, 175, 176
- configuration change record, 176
 - backing up, 176-177
 - restoring, 177
- configuration files, 88
- configuration files (*Continued*)
 - broker (figure), 88
 - cluster, 171, 172, 173, 283
 - default, 88
 - editing, 89-90
 - installation, 88
 - instance, 89, 172, 321, 322, 323
 - location, 321, 322, 323
 - template location, 321, 322, 323
 - templates, 321, 322, 323
- connecting brokers, 173
- connection factory administered objects
 - application server support attributes, 299
 - attributes, 156-162
 - client identification attributes, 159-160
 - connection handling attributes, 157-159
 - JMS properties support attributes, 162
 - overriding message header fields, 162
 - queue browser behavior attributes, 161, 298-299
 - reliability and flow control attributes, 161
 - standard message properties, 299-300
- connection service metrics
 - metric quantities, 312-313
 - using imqcmd metrics, 105-106, 188-189
 - using imqcmd query, 190
- connection services
 - access control for, 83, 274
 - activated at startup, 263
 - admin, 76, 103
 - cluster, 174
 - See* cluster connection service
 - commands affecting, 251-252
 - displaying properties of, 104
 - HTTP
 - See* HTTP connections
 - httpjms, 76, 103
 - HTTPS
 - See* HTTPS connections
 - httpsjms, 76, 103
 - See* httpsjms connection service
 - jms, 76, 103
 - metrics data
 - See* connection service metrics
 - pausing, 106, 251

connection services (*Continued*)

- Port Mapper
 - See* Port Mapper
 - properties, 105, 263-265
 - protocol type, 76
 - querying, 104, 108
 - resuming, 107, 251
 - service type, 76
 - SSL-based, 144
 - ssladmin
 - See* ssladmin connection service
 - ssljms
 - See* ssljms connection service
 - thread allocation, 105
 - thread pool management, 77
 - updating, 105, 108, 251
- connection services, broker, 75, 76
- connections
- automatic reconnection
 - See* automatic reconnection
 - failover
 - See* automatic reconnection
 - limited by file descriptor limits, 68
 - listing, 107, 252
 - performance effect of, 205-206
 - querying, 108, 252
 - server or broker failure, 348
- connectionURL Resource Adapter attribute, 304
- customAcknowledgeMode activation specification attribute, 307

D

data store

- about, 79
- compacting, 81
- configuring, 91
- contents of, 91
- flat-file, 80-81
- JDBC-compliant, 81-82
- location, 321, 322, 323
- performance effect of, 207
- resetting, 245
- synchronizing to disk, 91

dead message queue

- configuring, 123-125
 - limit behavior, 124
 - logging, 86, 125
 - maxNumMsgs value, 124
 - maxTotalMsgBytes value, 124
- dead messages
- See also* dead message queue
 - logging, 86
- default.properties file, 88
- deleting, broker instance, 72
- deleting destinations, 120-121
- delivery modes, performance effect of, 201
- destination activation specification attribute, 306
- destination administered objects, attributes, 162-163
- destination metrics
- metric quantities, 313-317
 - using imqcmd metrics, 187, 189
 - using imqcmd query, 190
 - using message-based monitoring, 191
- destinationType activation specification attribute, 306, 307
- destroying physical destinations, 120-121
- development environment administration tasks, 33-34
- directory lookup for clusters (Linux), 174
- disk space
- physical destination utilization, 121
 - reclaiming, 122
- displaying product version, 256
- distributed transactions, XA resource manager, 110
- durable subscriptions
- destroying, 109, 254
 - listing, 109, 254
 - managing, 108
 - performance effect of, 203
 - purging messages for, 254

E

encryption

- about, 82, 85
- Key Tool and, 85
- SSL-based services, 141-149

endpointExceptionRedeliveryAttempts activation
 specification attribute, 307, 308
 /etc/hosts file (Linux), 174
 example applications, 322, 323, 324

F

file-based persistence, 80-81
 file descriptor limits, 68
 connection limits and, 68
 file sync
 imq.persist.file.sync.enabled option, 271, 286
 with Sun Cluster, 271, 286
 firewalls, 329
 flow control, *See* message flow control
 fragmentation of messages, 81

G

guest user, 128

H

hardware, performance effect of, 205
 help (command line), 256
 hosts file (Linux), 174
 HTTP
 connection service
 See httpjms connection service
 proxy, 329
 support architecture, 329-330
 transport driver, 329
 HTTP connections
 multiple brokers, for, 337
 request interval, 335
 support for, 329
 tunnel servlet
 See HTTP tunnel servlet
 HTTP tunnel servlet
 about, 330
 deploying, 331-334

httpjms connection service
 about, 76, 103
 configuring, 334-336
 setting up, 331-337
 HTTPS
 connection service
 See httpsjms connection service
 support architecture, 329-330
 HTTPS connections
 multiple brokers, for, 347
 request interval, 344
 support for, 329
 tunnel servlet
 See HTTPS tunnel servlet
 HTTPS tunnel servlet
 about, 330
 deploying, 340-343
 httpsjms connection service, 141
 about, 76, 103
 configuring, 343-345
 setting up, 337-347

I

imq.accesscontrol.enabled property, 82, 274, 284
 imq.accesscontrol.file.filename property, 83, 275, 284
 imq.audit.enabled property, 85, 152, 278, 284
 imq.authentication.basic.user_repository property, 84, 276, 284
 imq.authentication.client.response.timeout property, 84, 276, 284
 imq.authentication.type property, 84, 275, 284
 imq.autocreate.destination.isLocalOnly property, 269, 284
 imq.autocreate.destination.limitBehavior property, 268, 284
 imq.autocreate.destination.maxBytesPerMsg property, 267, 284
 imq.autocreate.destination.maxCount property, 267, 284
 imq.autocreate.destination.maxNumMsgs property, 267, 284
 imq.autocreate.destination.maxNumProducers property, 268, 284

- imq.autocreate.destination.maxTotalMsgBytes property, 267, 269, 284
- imq.autocreate.destination.useDMQ property, 124
- imq.autocreate.queue.consumerFlowLimit property, 269, 284
- imq.autocreate.queue.localDeliveryPreferred property, 269, 284
- imq.autocreate.queue.maxNumActiveConsumers property, 99, 268, 284
- imq.autocreate.queue.maxNumBackupConsumers property, 99, 268, 284
- imq.autocreate.queue property, 99, 267, 284
- imq.autocreate.topic property, 99, 267, 284
- imq.cluster.brokerlist property, 171, 173, 174, 175, 176, 282, 284
- imq.cluster.masterbroker property, 171, 175, 176, 283, 284
- imq.cluster.port property, 172, 283, 284
- imq.cluster.transport property, 172, 174, 175, 283, 284
- imq.cluster.url property, 99, 171, 172, 173, 174, 176, 283, 284
- imq.destination.DMQ.truncateBody property, 79, 99, 266, 284
- imq.destination.logDeadMsgs property, 86, 99, 279, 285
- imq.hostname property, 77, 263, 285
- imq.httpjms.http.servletHost property, 335
- imq.imqcmd.password property, 84, 276, 285
- imq.keystore.file.dirpath property, 143, 278, 285
- imq.keystore.file.name property, 143, 285
- imq.keystore.password property, 85, 144, 151, 285
- imq.keystore.*property_name* property, 285
- imq.log.console.output property, 86, 279, 285
- imq.log.console.stream property, 86, 285
- imq.log.file.dirpath property, 86, 279, 285
- imq.log.file.filename property, 86, 279, 285
- imq.log.file.output property, 86, 280, 285
- imq.log.file.rolloverbytes property, 86, 99, 280, 285
- imq.log.file.rolloversecs property, 86, 99, 280, 285
- imq.log.level property, 86, 99, 278, 285
- imq.log.syslog.facility property, 281, 285
- imq.log.syslog.identity property, 281, 285
- imq.log.syslog.logconsole property, 281, 285
- imq.log.syslog.logpid property, 281, 285
- imq.log.syslog.output property, 86, 280, 285
- imq.log.timezone property, 281, 285
- imq.message.expiration.interval property, 78, 266, 285
- imq.message.max_size property, 78, 100, 266, 285
- imq.metrics.enabled property, 86, 281, 285
- imq.metrics.interval property, 86, 282, 285
- imq.metrics.topic.enabled property, 87, 282, 285
- imq.metrics.topic.interval property, 87, 282, 285
- imq.metrics.topic.persist property, 87, 282, 285
- imq.metrics.topic.timetolive property, 87, 282, 286
- imq.passfile.dirpath property, 84, 276, 286
- imq.passfile.enabled property, 84, 276, 286
- imq.passfile.name property, 84, 276, 286
- imq.persist.file.destination.message.filepool.limit property, 81, 271, 286
- imq.persist.file.message.cleanup property, 81, 271, 286
- imq.persist.file.message.filepool.cleanratio property, 81, 271, 286
- imq.persist.file.message.max_record_size property, 270, 286
- imq.persist.file.message.vrfile.max_record_size property, 81
- imq.persist.file.sync.enabled property, 81, 271, 286
 - Sun Cluster requirement, 271, 286
- imq.persist.file.sync property, 91
- imq.persist.jdbc.brokerid property, 82, 272, 286
- imq.persist.jdbc.closedburl property, 82, 272, 286
- imq.persist.jdbc.createdburl property, 82, 272, 286
- imq.persist.jdbc.driver property, 82, 272, 286
- imq.persist.jdbc.needpassword property, 82, 273, 286
- imq.persist.jdbc.opendburl property, 82, 272, 286
- imq.persist.jdbc.password property, 82, 151, 273, 286
- imq.persist.jdbc.table.IMQCCREC35 property, 82, 273, 286
- imq.persist.jdbc.table.IMQDEST35 property, 82, 273, 286
- imq.persist.jdbc.table.IMQINT35 property, 273, 286
- imq.persist.jdbc.table.IMQLIST35 property, 274, 286
- imq.persist.jdbc.table.IMQMSG35 property, 273, 286
- imq.persist.jdbc.table.IMQPROPS35 property, 273, 286
- imq.persist.jdbc.table.IMQSV35 property, 82, 273, 286
- imq.persist.jdbc.table.IMQTACK35 property, 274, 286
- imq.persist.jdbc.table.IMQTXN35 property, 274, 286

- imq.persist.jdbc.user property, 82, 272, 286
- imq.persist.store property, 80, 92, 270
- imq.ping.interval property, 78, 265, 286
- imq.portmapper.backlog property, 77, 264, 287
- imq.portmapper.hostname property, 77, 263, 287
- imq.portmapper.port property, 76, 100, 264, 287
- imq.protocol.protocolType.inbufsz, 209
- imq.protocol.protocolType.nodelay, 209
- imq.protocol.protocolType.outbufsz, 209
- imq.resource_state.count property, 266, 287
- imq.resource_state.threshold property, 266, 287
- imq.resourceState.count property, 79
- imq.service.activelist property, 76, 263, 287
- imq.service_name.accesscontrol.enabled property, 287
- imq.service_name.accesscontrol.file.filename property, 275, 287
- imq.service_name.authentication.type property, 275, 287
- imq.service_name.max_threads property, 265, 287
- imq.service_name.min_threads property, 264, 287
- imq.service_name.protocol_type.hostname property, 172, 264, 283, 284, 287
- imq.service_name.protocol_type.port property, 264, 287
- imq.service_name.threadpool_model property, 264, 287
- imq.serviceName.accesscontrol.enabled property, 83
- imq.serviceName.accesscontrol.file.filename property, 83
- imq.serviceName.authentication.type property, 84
- imq.serviceName.max_threads property, 78
- imq.serviceName.min_threads property, 78
- imq.serviceName.protocolType.hostname property, 77
- imq.serviceName.protocolType.port property, 77
- imq.serviceName.threadpool_model property, 77
- imq.shared.connectionMonitor_limit property, 78, 265, 287
- imq.system.max_count property, 78, 99, 265, 287
- imq.system.max_size property, 78, 100, 266, 287
- imq.transaction.autorollback property, 111, 266, 287
- imq.user_repository.ldap.base property, 277, 287
- imq.user_repository.ldap.gidattr property, 277, 287
- imq.user_repository.ldap.grpbase property, 277, 287
- imq.user_repository.ldap.grpfilter property, 277, 287
- imq.user_repository.ldap.grpsearch property, 277, 287
- imq.user_repository.ldap.memattr property, 277, 287
- imq.user_repository.ldap.password property, 84, 151, 277, 287
- imq.user_repository.ldap.principal property, 84, 277, 287
- imq.user_repository.ldap.property_name property, 277, 287
- imq.user_repository.ldap.server property, 84, 276, 287
- imq.user_repository.ldap.ssl.enabled property, 278, 288
- imq.user_repository.ldap.timeout property, 278, 288
- imq.user_repository.ldap.uidattr property, 277, 288
- imq.user_repository.ldap.usrfilter property, 277, 288
- imqAckTimeout attribute, 297
- imqAddressList attribute, 294
- imqAddressListBehavior attribute, 294
- imqAddressListIterations attribute, 294
- imqbrokerd command, 68
 - about, 36
 - adding a broker to a cluster, 174, 175
 - backing up configuration change record, 176
 - clearing the data store, 91, 120
 - configuration file (Solaris, Linux), 69, 73
 - connecting brokers, 173
 - in password file, 149
 - options, 244-248
 - passing arguments to, 90
 - reference, 244
 - removing a broker, 72
 - removing a broker from a cluster, 175
 - restoring configuration change record, 177
 - setting logging properties, 182
- imqbrokerd.conf file, 69, 73
- imqcmd command
 - about, 36
 - dependent on master broker, 177
 - durable subscription subcommands, 108
 - general options, 255, 260
 - in password file, 149
 - metrics monitoring, 186-190
 - physical destination management, 113
 - physical destination subcommands (table), 114
 - reference, 248

- imqcmd command (*Continued*)
 - secure connection to broker, 146, 255
 - transaction management, 110
 - imqConfiguredClientID attribute, 297
 - imqConnectionFlowCount attribute, 297
 - imqConnectionFlowLimit attribute, 298
 - imqConnectionFlowLimitEnabled attribute, 297
 - imqConsumerFlowLimit attribute, 298
 - imqConsumerFlowThreshold attribute, 298
 - imqdbmgr command
 - about, 36
 - in password file, 149
 - options, 258-259
 - reference, 257
 - imqDefaultPassword attribute, 297
 - imqDefaultUsername attribute, 297
 - imqDestinationDescription attribute, 301
 - imqDestinationName attribute, 301
 - imqDisableSetClientID attribute, 297
 - imqFlowControlLimit attribute, 298
 - imqJMSDeliveryMode attribute, 300
 - imqJMSExpiration attribute, 300
 - imqJMSPriority attribute, 162, 300
 - imqkeytool command
 - about, 36
 - command syntax, 142, 338
 - reference, 261
 - using, 142
 - imqLoadMaxToServerSession attribute, 161, 299
 - imqobjmgr command
 - about, 36
 - options, 256
 - reference, 256
 - subcommands, 256
 - imqOverrideJMSDeliveryMode attribute, 300
 - imqOverrideJMSExpiration attribute, 300
 - imqOverrideJMSHeadersToTemporaryDestinations attribute, 162, 300
 - imqOverrideJMSPriority attribute, 162, 300
 - imqQueueBrowserMax MessagesPerRetrieve attribute, 161, 299
 - imqQueueBrowserRetrieveTimeout attribute, 161, 299
 - imqReconnectAttempts attribute, 294
 - imqReconnectEnabled attribute, 294
 - imqReconnectInterval attribute, 294
 - imqSetJMSXAppID attribute, 299
 - imqSetJMSXConsumerTXID attribute, 300
 - imqSetJMSXProducerTXID attribute, 299
 - imqSetJMSXRcvTimestamp attribute, 300
 - imqSetJMSXUserID attribute, 299
 - imqSSLsHostTrusted attribute, 295
 - imqsvcadm command
 - about, 36
 - options, 260
 - reference, 260
 - subcommands, 260
 - imqusermgr command
 - about, 36
 - options, 259
 - passwords, 131
 - reference, 259
 - subcommands, 259
 - use for, 129
 - user names, 131
 - install.properties file, 88
 - instance configuration files, *See* configuration files
 - instance directory
 - and file-based data store, 91
 - and instance configuration file, 133
 - removing, 73
- ## J
- J2EE connector architecture (JCA), 303
 - java.naming.factory.initial attribute, 154, 155
 - java.naming.provider.url attribute, 154, 156
 - java.naming.security.authentication attribute, 155
 - java.naming.security.credentials attribute, 155
 - java.naming.security.principal attribute, 154
 - Java runtime
 - for Windows service, 71
 - specifying path to, 247, 256, 257, 261
 - Java Virtual Machine, *See* JVM
 - javahome option, 71
 - JCA (J2EE connector architecture), 303
 - JDBC-based persistence
 - about, 81-82
 - setting up, 91-93

JDBC-based persistence (*Continued*)

tuning for performance, 211

JDBC support

about, 81-82

configuring, 91

driver, 272

jms connection service, 76, 103

JMSDeliveryMode message header field, 162

JMSExpiration message header field, 162

JMSPriority message header field, 162

JNDI

lookup, 53

lookup name, 163, 164

object store, 36, 153

object store attributes, 154-155, 163

jrehome option, 71

JVM

metrics

See JVM metrics

performance effect of, 205

tuning for performance, 208-209

JVM metrics

metric quantities, 309-310

using broker log files, 185

using imqcmd metrics, 187

using message-based monitoring, 190

K

key pairs

generating, 143

regenerating, 144

key store

file, 143, 339

Key Tool, 85

L

LDAP server

as user repository, 133

authentication failover, 134

object store attributes, 154

user-repository access, 134

licenses, startup option, 247

limit behaviors

broker, 78

physical destinations, 115, 290

load-balanced queue delivery, tuning for

performance, 212-213

location of object store, 154

log files

changing default location, 181

changing default name, 181

dead message logging, 185-186

default location, 321, 322, 323

names, 181

reporting metrics, 184-185

rollover criteria, 86, 184, 280, 285

rollover frequency, 181

setting properties, 182

logger

about, 86-87

categories, 181

changing configuration, 182

dead message format, 185

levels, 86, 181, 247, 278, 285

message format, 182

metrics information, 281

output channels, 86, 181, 183-184

redirecting log messages, 183

rollover criteria, 184

setting properties, 182

writing to console, 86, 248, 279, 285

logging, *See* logger

loopback address, 174

M

ManagedConnectionFactory JavaBean, 305

master broker

configuration change record, 176

specifying, 171, 172

unavailable, 177

MDBs, *See* message-driven beans

memory management

for broker, 78

tuning for performance, 211-212

- memory management (*Continued*)
 - using physical destination properties, 115
 - message-driven beans
 - Resource Adapter configuration for, 303, 306
 - message expiration, clock synchronization and, 67
 - message flow control
 - attributes, 161
 - broker, 78, 115
 - limits, 213-215
 - metering, 213
 - performance effect of, 213
 - tuning for performance, 213
 - message header overrides, 162
 - message service architecture, 206-207
 - message service performance, 204-208
 - messages
 - body type and performance, 204
 - broker limits on, 78, 99, 265, 287
 - destination limits on, 267, 290
 - flow control
 - See* message flow control
 - fragmentation, 81
 - metrics messages
 - See* metrics messages
 - pausing flow of, 119
 - persistence of, 79
 - physical destination limits on, 115
 - purging from a physical destination, 120, 253
 - reclamation of expired, 78, 266, 285
 - reliable delivery of, 161
 - size, and performance, 203-204
 - messageSelector activation specification attribute, 306
 - metrics
 - about, 86
 - data
 - See* metrics data
 - messages
 - See* metrics messages
 - topic destinations, 87, 190-191
 - metrics data
 - broker
 - See* broker metrics
 - connection service
 - See* connection service metrics
 - metrics data (*Continued*)
 - physical destination
 - See* physical destination metrics
 - using broker log files, 184-185
 - using imqcmd metrics, 188
 - using message-based monitoring API, 191-192
 - metrics messages
 - about, 190
 - type, 87, 191
 - metrics monitoring tools
 - compared, 179-180
 - message-based monitoring API, 190-193
 - Message Queue Command Utility (imqcmd), 186-190
 - Message Queue log files, 184-185
 - monitoring, *See* performance monitoring
 - monitoring services, broker, 75, 85-87
- N**
- NORMAL service type, 76
 - nsswitch.conf file (Linux), 174
- O**
- object stores, 153-156
 - file-system, 155-156
 - file-system store attributes, 155
 - LDAP server, 153-155
 - LDAP server attributes, 154
 - locations, 321, 322, 323
 - operating system
 - performance effect of, 205
 - tuning Solaris performance, 208
 - Oracle, 92, 94
 - overrides
 - for message header, 162
 - on command line, 73

P

- password file
 - broker configuration properties, 84, 276
 - command line option, 246
 - location, 151, 322, 323, 324
 - using, 149-151
- password managed connection factory attribute, 305
- password Resource Adapter attribute, 304
- passwords
 - administrator, 132
 - default, 297
 - encoding of, 275
 - JDBC, 151
 - LDAP, 151
 - naming conventions, 131
 - password file
 - See* password file
 - SSL key store, 144, 151, 246
- pausing
 - brokers, 100-101, 250
 - connection services, 106, 251
 - physical destinations, 119, 253
- performance
 - about, 195-198
 - baseline patterns, 197-198
 - benchmarks, 196-197
 - bottlenecks, 200
 - factors affecting
 - See* performance factors
 - indicators, 196
 - measures of, 196
 - monitoring
 - See* performance monitoring
 - optimizing
 - See* performance tuning
 - reliability tradeoffs, 200
 - troubleshooting, 217
 - tuning
 - See* performance tuning
- performance factors
 - acknowledgment mode, 202
 - broker limit behaviors, 207
 - connections, 205-206
 - data store, 207
 - performance factors (*Continued*)
 - delivery mode, 201
 - durable subscriptions, 203
 - file sync, 271, 286
 - hardware, 205
 - JVM, 205
 - message body type, 204
 - message flow control, 213
 - message service architecture, 207
 - message size, 203-204
 - operating system, 205
 - selectors, 203
 - transactions, 201-202
 - transport protocols, 206
- performance monitoring
 - metrics data
 - See* metrics data
 - tools
 - See* metrics monitoring tools
- performance tuning
 - broker adjustments, 211-213
 - client runtime adjustments, 213-215
 - process overview, 195-196
 - system adjustments, 208-211
- permissions
 - access control properties file, 85, 136
 - admin service, 84
 - computing, 137-138
 - data store, 80
 - embedded database, 93
 - key store, 338
 - password file, 150
 - user repository, 129, 259
- persistence
 - about, 79
 - data store
 - See* data store
 - file-based, 80-81
 - JDBC
 - See* JDBC persistence
 - JDBC-based
 - See* JDBC-based persistence
 - options (figure), 80
 - properties, 270-271

- persistence (*Continued*)
 - security for, 93
 - persistence services, broker, 75, 79-82
 - physical destination
 - reclaiming disk space, 122
 - using dead message queue, 123
 - physical destinations
 - auto-created, 140
 - batching messages for delivery, 116, 269, 291
 - compacting, 121-123
 - compacting file-based data store, 122, 253
 - creating, 115-116
 - dead message queue, 123
 - dead message queue for, 123-125
 - destroying, 120-121
 - disk utilization, 121-123
 - displaying property values, 117-118
 - getting information about, 117, 254
 - information about, 117-118
 - limit behaviors, 115, 290
 - listing, 116-117
 - managing, 113
 - metrics
 - See physical destination metrics
 - pausing, 119, 253
 - properties of, 289-291
 - property values, 117
 - purging messages from, 120, 253
 - restricted scope in cluster, 116, 269, 291
 - resuming, 119
 - temporary, 117
 - types, 116, 252
 - updating attributes, 253
 - updating properties, 118
 - PointBase, 92
 - Port Mapper
 - about, 76
 - port assignment for, 244
 - precedence (of configuration properties), 88
 - producers
 - destination limits on, 268, 290
 - physical destination limits on, 116
 - production environment
 - administration tasks, 34-35
 - production environment (*Continued*)
 - maintaining, 35
 - setting up, 34-35
 - properties
 - auto-create, 267-269
 - broker instance configuration, 89
 - broker monitoring service, 278-282
 - cluster configuration, 282-283
 - connection services, 263-265
 - JDBC-related, 89, 272-274
 - logger, 278-282
 - memory management, 115
 - persistence, 270-271
 - physical destinations
 - See physical destinations, properties of
 - routine services, 265-267
 - security, 274-278
 - syntax, 90
 - protocol types
 - HTTP, 76, 103
 - TCP, 76, 103
 - TLS, 76, 103
 - protocols
 - See transport protocols
 - purging, messages from physical destinations, 120
- Q**
- querying
 - brokers, 98-99
 - connection services, 104, 108
 - queue load-balanced delivery
 - properties, 116, 268, 290, 291
 - queues
 - adding administered objects for, 165
 - auto-created, 267, 284
- R**
- reconnectAttempts managed connection factory
 - attribute, 305
 - reconnectAttempts Resource Adapter attribute, 304

- reconnectEnabled managed connection factory
 - attribute, 305
- reconnectEnabled Resource Adapter attribute, 304
- reconnectInterval managed connection factory
 - attribute, 305
- reconnectInterval Resource Adapter attribute, 304
- reconnection, automatic, *See* automatic reconnection
- reliable delivery, 161
 - performance tradeoffs, 200
- removing
 - brokers, 72
 - physical destinations, 120-121
- reset messages option, 120
- Resource Adapter, 303
 - reconnection, 304, 305
- ResourceAdapter JavaBean, 303
- RESTART property, 70
- restarting brokers, 101, 250
- resuming
 - brokers, 100-101, 101, 250
 - connection services, 107, 251
 - physical destinations, 119
- routine services, properties, 265-267
- routing services, broker, 75, 78

S

Secure Socket Layer standard, *See* SSL

security

- authentication
 - See* authentication
- authorization
 - See* authorization
- encryption
 - See* encryption
- manager
 - See* security manager
- object store, for, 154
- security manager
 - about, 82
 - properties, 274-278
- security services, broker, 75, 82-85
- selectors
 - about, 203

selectors (*Continued*)

- performance effect of, 203
- self-signed certificates, 141-146, 338
- sendUndeliverableMsgsToDMQ activation
 - specification attribute, 308
- server failure and secure connections, 348
- service (Windows)
 - Java runtime for, 71
 - reconfiguring, 70
 - removing broker, 73
 - running broker as, 70
 - startup parameters for, 71
 - troubleshooting startup, 72
- service types
 - ADMIN, 76
 - NORMAL, 76
- shutting down brokers, 101, 250
 - as Windows service, 73
- Simple Network Time Protocol, 67
- SNTP, 67
- SSL
 - about, 85
 - connection services
 - See* SSL-based connection services
 - enabling, 144
 - encryption and, 141-149
- SSL-based connection services
 - setting up, 141, 142
 - starting up, 145
- ssladmin connection service, 141
 - about, 76, 103
- ssljms connection service, 141
 - about, 76, 103
- starting
 - clients, 73
 - SSL-based connection services, 145
- startup parameters for broker Windows service, 71
- subscriptionDurability activation specification
 - attribute, 306, 307
- subscriptionName activation specification
 - attribute, 306, 307
- Sun Cluster, configuration for, 271
- synchronizing
 - clocks, 67

synchronizing (*Continued*)

- memory to disk, 91
- syntax for all commands, 243-244
- syslog, 86, 183
- system clock synchronization, 67

T

- TCP, 76, 103
- temporary physical destinations, 117
- thread pool management
 - about, 77
 - dedicated threads, 77
 - shared threads, 77
- time synchronization service, 67
- time-to-live, *See* message expiration
- TLS, 76, 103
- tools, administration, *See* administration tools
- topics
 - adding administered objects for, 165
 - auto-created, 267, 284
- transactions
 - committing, 111, 255
 - information about, 255
 - managing, 110-111
 - performance effect of, 201-202
 - rolling back, 110, 255
- transport protocols
 - performance effect of, 206
 - protocol types
 - See* protocol types
 - relative speeds, 206
 - tuning for performance, 209-211
- troubleshooting, 217
 - Windows service startup, 72
- tunnel servlet connection, 348
- tutorial, 39

U

- ulimit command, 68
- update dst subcommand, restrictions, 118

updating

- brokers, 99-100
- connection services, 105, 108, 251
- usage help, 256
- user groups, 130
 - default, 84
 - deleting assignment, 131
 - predefined, 130
- user names, 297
 - default, 128
 - format, 131
- user repository
 - about, 83
 - flat-file, 128
 - initial entries, 128
 - LDAP, 133
 - LDAP server, 134
 - location, 322, 323
 - managing, 131
 - platform dependence, 129, 259
 - populating, 131
 - user groups, 131
 - user states, 131
- userName managed connection factory attribute, 305
- userName Resource Adapter attribute, 304
- utilization ratio, 122

V

- version, 256

W

- W32Time service, 68
- Windows service, *See* service (Windows)
- write operations (for file based store), 91

X

- xntpd daemon, 67