



Sun Java System Message Queue 4.1 Developer's Guide for JMX Clients



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

Part No: 819-7758

Copyright 2007 Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95054 U.S.A. All rights reserved.

This product or document is protected by copyright and distributed under licenses restricting its use, copying, distribution, and decompilation. No part of this product or document may be reproduced in any form by any means without prior written authorization of Sun and its licensors, if any. Third-party software, including font technology, is copyrighted and licensed from Sun suppliers.

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, docs.sun.com, AnswerBook, AnswerBook2, 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 Sun™ 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.

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.

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 2007 Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95054 U.S.A. Tous droits réservés.

Ce produit ou document est protégé par un copyright et distribué avec des licences qui en restreignent l'utilisation, la copie, la distribution, et la décompilation. Aucune partie de ce produit ou document ne peut être reproduite sous aucune forme, par quelque moyen que ce soit, sans l'autorisation préalable et écrite de Sun et de ses bailleurs de licence, s'il y en a. Le logiciel détenu par des tiers, et qui comprend la technologie relative aux polices de caractères, est protégé par un copyright et licencié par des fournisseurs de Sun.

Des parties de ce produit pourront être dérivées du système Berkeley BSD licenciés par l'Université de Californie. UNIX est une marque déposée aux Etats-Unis et dans d'autres pays et licenciée exclusivement par X/Open Company, Ltd.

Sun, Sun Microsystems, le logo Sun, docs.sun.com, AnswerBook, AnswerBook2, 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.

CETTE PUBLICATION EST FOURNIE "EN L'ETAT" ET AUCUNE GARANTIE, EXPRESSE OU IMPLICITE, N'EST ACCORDEE, Y COMPRIS DES GARANTIES CONCERNANT LA VALEUR MARCHANDE, L'APTITUDE DE LA PUBLICATION A REpondre A UNE UTILISATION PARTICULIERE, OU LE FAIT QU'ELLE NE SOIT PAS CONTREFAISANTE DE PRODUIT DE TIERS. CE DENI DE GARANTIE NE S'APPLIQUERAIT PAS, DANS LA MESURE OU IL SERAIT TENU JURIDIQUEMENT NUL ET NON AVENU.

Contents

Preface	11
1 Introduction to JMX Programming for Message Queue Clients	19
JMX Architecture	19
Message Queue MBeans	20
Resource MBeans	20
Manager MBeans	21
Object Names	22
2 Using the JMX API	27
Interface Packages	27
Utility Classes	28
Connecting to the MBean Server	29
Obtaining a JMX Connector from an Administration Connection Factory	30
Obtaining a JMX Connector Without Using an Administration Connection Factory	31
JMX Service URLs	32
Using MBeans	33
Accessing MBean Attributes	33
Invoking MBean Operations	38
Receiving MBean Notifications	44
3 Message Queue MBean Reference	47
Message Brokers	47
Broker Configuration	47
Broker Monitor	51
Connection Services	53
Service Configuration	53

Service Monitor	55
Service Manager Configuration	58
Service Manager Monitor	59
Connections	61
Connection Configuration	61
Connection Monitor	62
Connection Manager Configuration	63
Connection Manager Monitor	64
Destinations	66
Destination Configuration	66
Destination Monitor	70
Destination Manager Configuration	75
Destination Manager Monitor	78
Message Producers	80
Producer Manager Configuration	80
Producer Manager Monitor	81
Message Consumers	83
Consumer Manager Configuration	84
Consumer Manager Monitor	85
Transactions	88
Transaction Manager Configuration	88
Transaction Manager Monitor	89
Broker Clusters	92
Cluster Configuration	92
Cluster Monitor	95
Logging	100
Log Configuration	100
Log Monitor	101
Java Virtual Machine	103
JVM Monitor	103
A Alphabetical Reference	105
Index	117

Tables

TABLE 1-1	Object Name Properties	23
TABLE 1-2	Message Queue MBean Types	23
TABLE 1-3	Message Queue MBean Subtypes	24
TABLE 1-4	Destination Types	24
TABLE 1-5	Connection Service Names	24
TABLE 1-6	Example Object Names	25
TABLE 1-7	Utility Constants and Methods for Object Names	26
TABLE 2-1	JMX . jar File Locations	27
TABLE 2-2	Message Queue JMX Utility Classes	28
TABLE 3-1	Broker Configuration Attributes	48
TABLE 3-2	Broker Configuration Operations	49
TABLE 3-3	Broker Configuration Notification	51
TABLE 3-4	Broker Monitor Attributes	51
TABLE 3-5	Broker Monitor Notifications	52
TABLE 3-6	Data Retrieval Methods for Broker Monitor Notifications	52
TABLE 3-7	Connection Service Names for Service Configuration MBeans	53
TABLE 3-8	Service Configuration Attributes	54
TABLE 3-9	Service Configuration Operations	54
TABLE 3-10	Service Configuration Notification	55
TABLE 3-11	Connection Service Names for Service Monitor MBeans	55
TABLE 3-12	Service Monitor Attributes	56
TABLE 3-13	Connection Service State Values	57
TABLE 3-14	Service Monitor Operations	57
TABLE 3-15	Service Monitor Notifications	57
TABLE 3-16	Data Retrieval Method for Service Monitor Notifications	58
TABLE 3-17	Service Manager Configuration Attributes	58
TABLE 3-18	Service Manager Configuration Operations	59
TABLE 3-19	Service Manager Monitor Attributes	59

TABLE 3-20	Service Manager Monitor Operation	60
TABLE 3-21	Service Manager Monitor Notifications	60
TABLE 3-22	Data Retrieval Method for Service Manager Monitor Notifications	61
TABLE 3-23	Connection Configuration Attribute	62
TABLE 3-24	Connection Monitor Attributes	62
TABLE 3-25	Connection Monitor Operations	63
TABLE 3-26	Connection Manager Configuration Attribute	64
TABLE 3-27	Connection Manager Configuration Operations	64
TABLE 3-28	Connection Manager Monitor Attributes	65
TABLE 3-29	Connection Manager Monitor Operation	65
TABLE 3-30	Connection Manager Monitor Notifications	65
TABLE 3-31	Data Retrieval Methods for Connection Manager Monitor Notifications	65
TABLE 3-32	Destination Configuration Attributes	67
TABLE 3-33	Destination Configuration Type Values	68
TABLE 3-34	Destination Limit Behaviors	68
TABLE 3-35	Destination Configuration Operations	69
TABLE 3-36	Destination Pause Types	69
TABLE 3-37	Destination Configuration Notification	70
TABLE 3-38	Destination Monitor Attributes	70
TABLE 3-39	Destination Monitor Type Values	73
TABLE 3-40	Destination State Values	73
TABLE 3-41	Destination Monitor Operations	74
TABLE 3-42	Destination Monitor Notifications	74
TABLE 3-43	Data Retrieval Methods for Destination Monitor Notifications	75
TABLE 3-44	Destination Manager Configuration Attributes	75
TABLE 3-45	Destination Manager Configuration Operations	77
TABLE 3-46	Destination Manager Configuration Type Values	78
TABLE 3-47	Destination Manager Pause Types	78
TABLE 3-48	Destination Manager Configuration Notification	78
TABLE 3-49	Destination Manager Monitor Attributes	79
TABLE 3-50	Destination Manager Monitor Operation	79
TABLE 3-51	Destination Manager Monitor Notifications	79
TABLE 3-52	Data Retrieval Methods for Destination Manager Monitor Notifications	80
TABLE 3-53	Producer Manager Configuration Attribute	81
TABLE 3-54	Producer Manager Configuration Operation	81
TABLE 3-55	Producer Manager Monitor Attribute	82

TABLE 3-56	Producer Manager Monitor Operations	82
TABLE 3-57	Lookup Keys for Message Producer Information	83
TABLE 3-58	Message Producer Destination Types	83
TABLE 3-59	Consumer Manager Configuration Attribute	84
TABLE 3-60	Consumer Manager Configuration Operations	84
TABLE 3-61	Consumer Manager Monitor Attribute	85
TABLE 3-62	Consumer Manager Monitor Operations	85
TABLE 3-63	Lookup Keys for Message Consumer Information	86
TABLE 3-64	Message Consumer Destination Types	87
TABLE 3-65	Acknowledgment Modes	87
TABLE 3-66	Transaction Manager Configuration Attribute	88
TABLE 3-67	Transaction Manager Configuration Operations	89
TABLE 3-68	Transaction Manager Monitor Attributes	89
TABLE 3-69	Transaction Manager Monitor Operations	90
TABLE 3-70	Lookup Keys for Transaction Information	90
TABLE 3-71	Transaction State Values	91
TABLE 3-72	Transaction Manager Monitor Notifications	92
TABLE 3-73	Data Retrieval Method for Transaction Manager Monitor Notifications	92
TABLE 3-74	Cluster Configuration Attributes	93
TABLE 3-75	Cluster Configuration Operations	94
TABLE 3-76	Lookup Keys for Cluster Configuration Information	95
TABLE 3-77	Cluster Configuration Notification	95
TABLE 3-78	Cluster Monitor Attributes	96
TABLE 3-79	Cluster Monitor Operations	97
TABLE 3-80	Lookup Keys for Cluster Monitor Information	98
TABLE 3-81	Broker State Values	98
TABLE 3-82	Cluster Monitor Notifications	99
TABLE 3-83	Data Retrieval Methods for Cluster Monitor Notifications	100
TABLE 3-84	Log Configuration Attributes	101
TABLE 3-85	Log Configuration Logging Levels	101
TABLE 3-86	Log Configuration Notification	101
TABLE 3-87	Log Monitor Notifications	102
TABLE 3-88	Data Retrieval Methods for Log Monitor Notifications	102
TABLE 3-89	JVM Monitor Attributes	103
TABLE A-1	Alphabetical List of MBean Attributes	105
TABLE A-2	Alphabetical List of MBean Operations	110

TABLE A-3	Alphabetical List of MBean Notifications	113
-----------	--	-----

Examples

EXAMPLE 2-1	Obtaining a JMX Connector from an Administration Connection Factory	30
EXAMPLE 2-2	Configuring an Administration Connection Factory	31
EXAMPLE 2-3	Obtaining a JMX Connector Without Using an Administration Connection Factory	31
EXAMPLE 2-4	Getting an Attribute Value	33
EXAMPLE 2-5	Getting Multiple Attribute Values	34
EXAMPLE 2-6	Setting an Attribute Value	35
EXAMPLE 2-7	Setting Multiple Attribute Values	37
EXAMPLE 2-8	Invoking an Operation	38
EXAMPLE 2-9	Invoking an Operation with Parameters	39
EXAMPLE 2-10	Combining Operations and Attributes	41
EXAMPLE 2-11	Using a Composite Data Object	42
EXAMPLE 2-12	Notification Listener	45
EXAMPLE 2-13	Registering a Notification Listener	45

Preface

This *Message Queue Developer's Guide for JMX Clients* describes the application programming interface provided in Sun Java System Message Queue 4.1 for programmatically configuring and monitoring Message Queue resources in conformance with the Java Management Extensions (JMX). As in earlier versions of Message Queue, these functions are also available to system administrators by way of the Message Queue Administration Console and command line utilities, as described in the *Message Queue Administration Guide*. As of release 4.1, the API described here makes the same administrative functionality available programmatically from within a running client application.

Message Queue 4.1 also includes several new broker properties and command-line options to support the new JMX API. These features are described in the *Message Queue Release Notes* for release 4.1, and will eventually be incorporated into the *Message Queue Administration Guide*.

Who Should Use This Book

This guide is intended for Java application developers wishing to use the Message Queue JMX API to perform Message Queue administrative tasks programmatically from within a client application.

Before You Read This Book

This guide assumes that you are already familiar with general Message Queue concepts, administrative operations, and Java client programming, as described in the following manuals:

- *Message Queue Technical Overview*
- *Message Queue Administration Guide*
- *Message Queue Developer's Guide for Java Clients*

You should also be familiar with the general principles of the Java Management Extensions, as described in the following publications:

- *Java Management Extensions Instrumentation and Agent Specification*
- *Java Management Extensions (JMX) Remote API Specification*

Together, these two publications are referred to hereafter as the *JMX Specification*.

How This Book Is Organized

This guide consists of the following chapters:

- [Chapter 1, “Introduction to JMX Programming for Message Queue Clients”](#) introduces the basic concepts and principles of the Message Queue JMX interface.
- [Chapter 2, “Using the JMX API”](#) provides code examples showing how to use the JMX application programming interface from within your Message Queue client applications.
- [Chapter 3, “Message Queue MBean Reference”](#) provides detailed information on the attributes, operations, and notifications provided by Message Queue managed beans (MBeans).
- [Appendix A, “Alphabetical Reference”](#) lists the MBean attributes, operations, and notifications alphabetically, with references back to their descriptions in the body of the manual.

Related Documentation

In addition to this guide, Sun provides the additional documentation resources described in the following subsections.

Message Queue Documentation Set

The Message Queue documentation set comprises the documents shown in [Table P-1](#), in the order in which you would normally use them.

TABLE P-1 Message Queue Documentation Set

Document	Audience	Description
<i>Message Queue Installation Guide</i>	Developers and administrators	Explains how to install Message Queue software on Solaris, Linux, and Windows platforms
<i>Message Queue Release Notes</i>	Developers and administrators	Includes descriptions of new features, limitations, and known bugs, as well as technical notes
<i>Message Queue Technical Overview</i>	Developers and administrators	Introduces basic Message Queue concepts, features, and components
<i>Message Queue Administration Guide</i>	Administrators (also recommended for developers)	Provides background and information needed to perform administrative tasks using Message Queue administration tools

TABLE P-1 Message Queue Documentation Set (Continued)

Document	Audience	Description
<i>Message Queue Developer's Guide for Java Clients</i>	Developers	Provides information on developing Java client programs using the Message Queue implementation of the Java Message Service (JMS) and SOAP/JAXM specifications
<i>Message Queue Developer's Guide for C Clients</i>	Developers	Provides information on developing C and C++ client programs using Message Queue's C application programming interface (C API)
<i>Message Queue Developer's Guide for JMX Clients</i>	Developers	Provides information on developing Java client programs using the Message Queue implementation of the Java Management Extensions (JMX) API

Java Management Extensions (JMX) Documentation

The Message Queue JMX API conforms to the Java Management Extensions (JMX) standard, described in the *Java Management Extensions Instrumentation and Agent Specification* and the *Java Management Extensions (JMX) Remote API Specification*. These documents can be downloaded from the URLs

<http://jcp.org/aboutJava/communityprocess/final/jsr003>

and

<http://jcp.org/aboutJava/communityprocess/final/jsr160>

respectively.

For a general conceptual introduction to JMX principles and architecture, see the *Java Management Extensions (JMX) Technology Overview* at

<http://java.sun.com/j2se/1.5.0/docs/guide/jmx/overview/JMXoverviewTOC.html>

and the *Java Management Extensions (JMX) Technology Tutorial* at

<http://java.sun.com/j2se/1.5.0/docs/guide/jmx/tutorial/tutorialTOC.html>

JavaDoc

Message Queue API documentation in JavaDoc format is included in your Message Queue installation at the locations shown in [Table P-2](#), depending on your platform. 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.

TABLE P-2 JavaDoc Locations

Platform	Location
Solaris	/usr/share/javadoc/imq/index.html
Linux	/opt/sun/mq/javadoc/index.html
Windows	IMQ_HOME\javadoc\index.html where IMQ_HOME is the Message Queue base directory set by the Message Queue Installer (C:\Program Files\Sun\MessageQueue4 by default)

Example Client Applications

Example client applications providing sample Java application code using JMX are included in your Message Queue installation at the locations shown in [Table P-3](#), depending on your platform.

TABLE P-3 JMX Code Example Locations

Platform	Location
Solaris	/usr/demo/imq/jmx
Linux	/opt/sun/mq/examples/jmx
Windows	IMQ_HOME\demo\jmx where IMQ_HOME is the Message Queue base directory set by the Message Queue installer (C:\Program Files\Sun\MessageQueue4 by default)

Typographic Conventions

[Table P-4](#) shows the typographic conventions used in Message Queue documentation.

TABLE P-4 Typographic Conventions

Typeface	Meaning	Example
AaBbCc123	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, as contrasted with onscreen computer output	<code>machine_name% su</code> Password:

TABLE P-4 Typographic Conventions (Continued)

Typeface	Meaning	Example
<i>aabbcc123</i>	Placeholder: replace with a real name or value	The command to remove a file is <code>rm filename</code> .
<i>AaBbCc123</i>	Book titles, new terms, and emphasized words	Read Chapter 6 in the <i>User's Guide</i> . A <i>cache</i> is a copy that is stored locally. Do <i>not</i> save the file. Note – Some emphasized items appear online in boldface .

Symbol Conventions

Table P-5 shows symbol conventions used in Message Queue documentation.

TABLE P-5 Symbol Conventions

Symbol	Description	Example	Meaning
[]	Encloses optional arguments and command options	<code>ls [-l]</code>	The <code>-l</code> option is optional.
{ }	Encloses 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.
`\${ }`	Indicates a variable reference	<code>\${com.sun.javaRoot}</code>	References the value of the variable <code>com.sun.javaRoot</code> .
-	Joins simultaneous multiple keystrokes	Ctrl-A	Hold down the Control key while pressing the A key.
+	Joins consecutive multiple keystrokes	Ctrl+A+N	Press the Control key, release it, and then press the subsequent keys.
→	Indicates hierarchical menu selection in a graphical user interface	File → New → Templates	From the File menu, choose New; from the New submenu, choose Templates.

Directory Variable Conventions

Message Queue makes use of three directory variables; how they are set varies from platform to platform. [Table P-6](#) describes these variables and how they are used on the Solaris, Linux, and Windows platforms.

Note – The information in [Table P-6](#) 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` is set to `appServer_domainName_dir/imq` (where `appServer_domainName_dir` is the domain directory for the domain starting the Message Queue broker).

TABLE P-6 Directory Variable Conventions

Variable	Description
<code>IMQ_HOME</code>	<p>The Message Queue base directory (root installation directory):</p> <ul style="list-style-type: none"> ■ Unused on Solaris and Linux; there is no Message Queue base directory. ■ On Windows, set by the Message Queue Installer to the directory in which you unzip the Message Queue bundle.
<code>IMQ_VARHOME</code>	<p>The directory in which Message Queue temporary or dynamically created configuration and data files are stored; can be set as an environment variable to point to any directory.</p> <ul style="list-style-type: none"> ■ On Solaris, defaults to <code>/var/imq</code>. ■ On Linux, defaults to <code>/var/opt/sun/mq</code>. ■ On Windows, defaults to <code>IMQ_HOME\var</code>.
<code>IMQ_JAVAHOME</code>	<p>The location of the Java runtime environment (JRE) required by Message Queue executables:</p> <ul style="list-style-type: none"> ■ On Solaris and Linux, set by default to the location of the latest JRE, but can optionally be set to point to another, preferred JRE instead. ■ On Windows, set to the location of an existing JRE if a supported version is found on the system. If a supported version is not found, one will be installed.

Note – In this manual, these directory variables are shown without platform-specific environment variable notation or syntax (such as `$IMQ_HOME` on UNIX). Pathnames generally use UNIX directory separator notation (`/`).

Shell Prompts in Command Examples

Table P-7 shows the default UNIX® system prompt and superuser prompt for the C shell, Bourne shell, Korn shell, and Windows operating system.

TABLE P-7 Shell Prompts

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

Documentation, Support, and Training

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

- [Documentation](http://www.sun.com/documentation/) (<http://www.sun.com/documentation/>)
- [Support](http://www.sun.com/support/) (<http://www.sun.com/support/>)
- [Training](http://www.sun.com/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 such as 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 referenced in this document 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 always interested in improving its documentation and welcomes your comments and suggestions. To share your comments, go to the Sun documentation Web site at

<http://docs.sun.com>

and click Send Comments. In the resulting online form, provide the document title and part number along with your comment. (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 819-7758.)

Introduction to JMX Programming for Message Queue Clients

While Sun Java System Message Queue's Administration Console and command line administration utilities allow an administrator to interactively configure and monitor Message Queue resources (such as message brokers, connections, and destinations), these tools are not accessible from within a running client application. To provide programmatic access to such administrative functions, Message Queue also incorporates an application programming interface based on the *Java Management Extensions (JMX)*. Client applications can use this API to perform the same configuration and monitoring operations programmatically that are available interactively through the Administration Console and command line utilities.

You can use Message Queue's JMX API in your client applications for a variety of purposes:

- To optimize performance by monitoring the usage of message brokers and other Message Queue resources and reconfiguring their parameters based on the results
- To automate regular maintenance tasks, rolling upgrades, and so forth
- To write your own utility applications to replace or enhance standard Message Queue tools such as the Broker utility (`imqbrokerd`) and Command utility (`imqcmd`)

In addition, since JMX is the Java standard for building management applications and is widely used for managing J2EE infrastructure, you can use it to incorporate your Message Queue client as part of a larger J2EE deployment using a standard management framework throughout.

JMX Architecture

The JMX Specification defines an architecture for the instrumentation and programmatic management of distributed resources. This architecture is based on the notion of a *managed bean*, or *MBean*: a Java object, similar to a *JavaBean*, representing a resource to be managed. Message Queue MBeans may be associated with individual resources such as message brokers or destinations, or with whole categories of resources, such as the set of all destinations on a broker. There are separate *configuration MBeans* and *monitor MBeans* for setting a resource's configuration properties and monitoring its runtime state.

Each MBean is identified by an *object name*, an instance of the JMX class `ObjectName` conforming to the syntax and conventions defined in the JMX Specification. Object names for Message Queue MBeans are either defined as static constants or returned by static methods in the Message Queue utility class `MQObjectName`; see “Object Names” on page 22 for further information.

An MBean provides access to its underlying resource through a management interface consisting of the following:

- *Attributes* holding data values representing static or dynamic properties of the resource
- *Operations* that can be invoked to perform actions on the resource
- *Notifications* informing the client application of state changes or other significant events affecting the resource

Client applications obtain MBeans through an *MBean server*, which serves as a container and registry for MBeans. Each Message Queue broker process contains an MBean server, accessed by means of a *JMX connector*. The JMX connector is used to obtain an *MBean server connection*, which in turn provides access to individual MBeans on the server. Configuring or monitoring a Message Queue resource with JMX requires the following steps:

1. Obtain a JMX connector.
2. Get an MBean server connection from the JMX connector.
3. Construct an object name identifying the particular MBean you wish to operate on.
4. Pass the object name to the appropriate methods of the MBean server connection to access the MBean’s attributes, operations, and notifications.
5. Close the MBean server connection.

See [Chapter 2, “Using the JMX API”](#) for code examples illustrating the technique for various MBean operations.

Message Queue MBeans

Message Queue’s JMX functionality is exposed through MBeans associated with various Message Queue resources. These MBeans are of two kinds: *resource MBeans* and *manager MBeans*. The attributes, operations, and notifications available for each type of MBean are described in detail in [Chapter 3, “Message Queue MBean Reference.”](#)

Resource MBeans

Resource MBeans are associated with individual Message Queue resources of the following types:

- Message brokers

- Connection services
- Connections
- Destinations
- Broker clusters
- Logging
- The Java Virtual Machine (JVM)

Configuration and monitoring functions are implemented by separate MBeans. Each managed resource is associated with a *configuration MBean* for setting the resource's configuration and a *monitor MBean* for gathering (typically transient) information about its runtime state. For instance, there is a destination configuration MBean for configuring a destination and a destination monitor MBean for obtaining runtime information about it. In general, each instance of a managed resource has its own pair of MBeans: thus there is a separate destination configuration MBean and destination monitor MBean for each individual destination. (In the case of the Java Virtual Machine, there is only a JVM monitor MBean with no corresponding configuration MBean.)

Configuration MBeans are used to perform such tasks as the following:

- Set a broker's port number
- Set a broker's maximum message size
- Pause a connection service
- Set the maximum number of threads for a connection service
- Purge all messages from a destination
- Set the level of logging information to be written to an output channel

Monitor MBeans are used to obtain runtime information such as the following:

- The current number of connections on a service
- The cumulative number of messages received by a destination since the broker was started
- The current state (running or paused) of a queue destination
- The current number of message producers for a topic destination
- The host name and port number of a cluster's master broker
- The current JVM heap size

Manager MBeans

In addition to the resource MBeans associated with individual resources, there are also *manager MBeans* for managing some whole categories of resources. These manager MBeans also come in pairs—one for configuration and one for monitoring—for the following resource categories:

- Connection services
- Connections
- Destinations

- Message producers
- Message consumers
- Transactions

Unlike individual resource MBeans, a broker has only one pair of manager MBeans for each whole category of resources: for instance, a single destination manager configuration MBean and a single destination manager monitor MBean. For some categories (connection services, connections, destinations), the manager MBeans exist in addition to the ones for individual resources, and are used to manage the collection of resource MBeans within the category or to perform global tasks that are beyond the scope of individual resource MBeans. Thus, for instance, there is a connection manager configuration MBean and a connection manager monitor MBean in addition to the connection configuration and connection monitor MBeans associated with individual connections. Manager MBeans of this type are used to perform tasks such as the following:

- Get the object names of the service monitor MBeans for all available connection services
- Get the total number of current connections
- Destroy a connection
- Create or destroy a destination
- Enable or disable auto-creation of destinations
- Pause message delivery for all destinations

In other cases (message producers, message consumers, transactions), there are no MBeans associated with individual resources and all of the resources in the category are managed through the manager MBeans themselves. The manager MBeans for these categories can be used for such tasks as the following:

- Get the destination name associated with a message producer
- Purge all messages from a durable subscriber
- Commit or roll back a transaction

Object Names

Each individual MBean is designated by an *object name* belonging to the JMX class `ObjectName`, which encapsulates a string identifying the MBean. For Message Queue MBeans, the encapsulated name string has the following syntax:

```
com.sun.messaging.jms.server:property=value[,property=value]*
```

Table 1–1 shows the possible properties.

TABLE 1-1 Object Name Properties

Property	Description	Values
type	MBean type	See Table 1-2 .
subtype	MBean subtype	See Table 1-3 .
dest type	Destination type Applies only to MBeans of the following types: <ul style="list-style-type: none"> Destination configuration Destination monitor 	See Table 1-4 .
name	Resource name Applies only to MBeans of the following types: <ul style="list-style-type: none"> Service configuration Service monitor Destination configuration Destination monitor 	For service configuration and service monitor MBeans, see Table 1-5 . For destination configuration and destination monitor MBeans, the destination name. Examples: myTopic temporary_destination://queue/129.145.180.99/63008/1
id	Resource identifier Applies only to MBeans of the following types: <ul style="list-style-type: none"> Connection configuration Connection monitor 	Example: 7853717387765338368

[Table 1-2](#) shows the possible values for the object name's type property.

TABLE 1-2 Message Queue MBean Types

Value	Description
Broker	Broker resource MBean
Service	Connection service resource MBean
ServiceManager	Connection service manager MBean
Connection	Connection resource MBean
ConnectionManager	Connection manager MBean
Destination	Destination resource MBean
DestinationManager	Destination manager MBean
ProducerManager	Message producer manager MBean

TABLE 1-2 Message Queue MBean Types (Continued)

Value	Description
ConsumerManager	Message consumer manager MBean
TransactionManager	Transaction manager MBean
Cluster	Broker cluster resource MBean
Log	Logging resource MBean
JVM	JVM resource MBean

Table 1-3 shows the possible values for the object name's subtype property.

TABLE 1-3 Message Queue MBean Subtypes

Value	Description
Config	Configuration MBean
Monitor	Monitor MBean

For destination configuration and destination monitor MBeans, the object name's `destType` property specifies whether the destination is a point-to-point queue or a publish/subscribe topic. Table 1-4 shows the possible values, which are defined for convenience as static constants in the utility class `DestinationType`.

TABLE 1-4 Destination Types

Value	Utility Constant	Meaning
q	<code>DestinationType.QUEUE</code>	Queue (point-to-point) destination
t	<code>DestinationType.TOPIC</code>	Topic (publish/subscribe) destination

For service configuration and service monitor MBeans, the object name's `name` property identifies the connection service with which the MBean is associated. Table 1-5 shows the possible values.

TABLE 1-5 Connection Service Names

Service Name	Service Type	Protocol Type
jms	Normal	TCP
ssljms	Normal	TLS (SSL-based security)
httpjms	Normal	HTTP

TABLE 1-5 Connection Service Names (Continued)

Service Name	Service Type	Protocol Type
httpsjms	Normal	HTTPS (SSL-based security)
admin	Admin	TCP
ssladmin	Admin	TLS (SSL-based security)

Table 1-6 shows some example object names.

TABLE 1-6 Example Object Names

MBean type	Object Name
Broker configuration	com.sun.messaging.jms.server:type=Broker,subtype=Config
Service manager monitor	com.sun.messaging.jms.server:type=ServiceManager,subtype=Monitor
Connection configuration	com.sun.messaging.jms.server:type=Connection,subtype=Config,id=7853717387765338368
Destination monitor	com.sun.messaging.jms.server:type=Destination,subtype=Monitor,desttype=t,name="MyQueue"

The object names for each type of Message Queue MBean are given in the relevant sections of Chapter 3, “Message Queue MBean Reference.” All such names are either defined as static constants or returned by static methods in the utility class MQObjectName (see Table 1-7). For instance, the constant

```
MQObjectName.BROKER_CONFIG_MBEAN_NAME
```

is defined as a string representing the object name for a broker configuration MBean, and the method call

```
MQObjectName.createDestinationMonitor(DestinationType.TOPIC, "MyQueue");
```

returns the destination monitor MBean object name shown in Table 1-6. Note that, whereas methods such as createDestinationMonitor return an actual object name (that is, an object of class ObjectName) that can be assigned directly to a variable of that type

```
ObjectName destMonitorName
    = MQObjectName.createDestinationMonitor(DestinationType.TOPIC, "Dest");
```

constants like BROKER_CONFIG_MBEAN_NAME instead represent an ordinary string (class String) that must then be converted into the corresponding object name itself:

```
ObjectName brokerConfigName
    = new ObjectName(MQObjectName.BROKER_CONFIG_MBEAN_NAME);
```

TABLE 1-7 Utility Constants and Methods for Object Names

MBean Type	Utility Constant or Method
Broker configuration	<code>MQObjectName.BROKER_CONFIG_MBEAN_NAME</code>
Broker monitor	<code>MQObjectName.BROKER_MONITOR_MBEAN_NAME</code>
Service configuration	<code>MQObjectName.createServiceConfig</code>
Service monitor	<code>MQObjectName.createServiceMonitor</code>
Service manager configuration	<code>MQObjectName.SERVICE_MANAGER_CONFIG_MBEAN_NAME</code>
Service manager monitor	<code>MQObjectName.SERVICE_MANAGER_MONITOR_MBEAN_NAME</code>
Connection configuration	<code>MQObjectName.createConnectionConfig</code>
Connection monitor	<code>MQObjectName.createConnectionMonitor</code>
Connection manager configuration	<code>MQObjectName.CONNECTION_MANAGER_CONFIG_MBEAN_NAME</code>
Connection manager monitor	<code>MQObjectName.CONNECTION_MANAGER_MONITOR_MBEAN_NAME</code>
Destination configuration	<code>MQObjectName.createDestinationConfig</code>
Destination monitor	<code>MQObjectName.createDestinationMonitor</code>
Destination manager configuration	<code>MQObjectName.DESTINATION_MANAGER_CONFIG_MBEAN_NAME</code>
Destination manager monitor	<code>MQObjectName.DESTINATION_MANAGER_MONITOR_MBEAN_NAME</code>
Producer manager configuration	<code>MQObjectName.PRODUCER_MANAGER_CONFIG_MBEAN_NAME</code>
Producer manager monitor	<code>MQObjectName.PRODUCER_MANAGER_MONITOR_MBEAN_NAME</code>
Consumer manager configuration	<code>MQObjectName.CONSUMER_MANAGER_CONFIG_MBEAN_NAME</code>
Consumer manager monitor	<code>MQObjectName.CONSUMER_MANAGER_MONITOR_MBEAN_NAME</code>
Transaction manager configuration	<code>MQObjectName.TRANSACTION_MANAGER_CONFIG_MBEAN_NAME</code>
Transaction manager monitor	<code>MQObjectName.TRANSACTION_MANAGER_MONITOR_MBEAN_NAME</code>
Cluster configuration	<code>MQObjectName.CLUSTER_CONFIG_MBEAN_NAME</code>
Cluster monitor	<code>MQObjectName.CLUSTER_MONITOR_MBEAN_NAME</code>
Log configuration	<code>MQObjectName.LOG_CONFIG_MBEAN_NAME</code>
Log monitor	<code>MQObjectName.LOG_MONITOR_MBEAN_NAME</code>
JVM monitor	<code>MQObjectName.JVM_MONITOR_MBEAN_NAME</code>

Using the JMX API

This chapter provides code examples showing how to use the JMX application programming interface to connect to a message broker's MBean server, obtain MBeans for Message Queue resources, and access their attributes, operations, and notifications.

Interface Packages

The Message Queue 4.1 installation includes two Java packages related to the JMX interface:

- `com.sun.messaging` contains the class `AdminConnectionFactory` (discussed in “[Connecting to the MBean Server](#)” on page 29), along with a utility class `AdminConnectionFactoryConfiguration` defining static constants for use in configuring it.
- `com.sun.messaging.jms.management.server` contains a collection of utility classes (listed in “[Utility Classes](#)” on page 28) defining useful static constants and methods used in the JMX interface.

These packages are contained in a Java archive file, `imqjmx.jar`, included in your Message Queue installation at the locations shown in [Table 2-1](#), depending on your platform.

TABLE 2-1 JMX.jar File Locations

Platform	File Location
Solaris	<code>/usr/share/lib/imqjmx.jar</code>
Linux	<code>/opt/sun/mq/share/lib/imqjmx.jar</code>
Solaris	<code>C:\sun\lib\imqjmx.jar</code>

To do application development for the Message Queue JMX API, you must include this `.jar` file in your `CLASSPATH` environment variable.

Note – Message Queue’s JMX interface requires version 1.5 of the Java Development Kit (JDK). The functionality described here is not available under earlier versions of the JDK.

Utility Classes

The package `com.sun.messaging.jms.management.server` in the Message Queue JMX interface contains a collection of utility classes defining useful static constants and methods for use with Message Queue MBeans. [Table 2–2](#) lists these utility classes; see the relevant sections of [Chapter 3, “Message Queue MBean Reference”](#) and the Message Queue JMX JavaDoc documentation for further details.

TABLE 2–2 Message Queue JMX Utility Classes

Class	Description
<code>MQObjectName</code>	Constants and methods for Message Queue MBean object names
<code>MQNotification</code>	Superclass for all Message Queue JMX notifications
<code>BrokerAttributes</code>	Names of broker attributes
<code>BrokerOperations</code>	Names of broker operations
<code>BrokerNotification</code>	Constants and methods related to broker notifications
<code>BrokerState</code>	Constants related to broker state
<code>ServiceAttributes</code>	Names of connection service attributes
<code>ServiceOperations</code>	Names of connection service operations
<code>ServiceNotification</code>	Constants and methods related to connection service notifications
<code>ServiceState</code>	Constants related to connection service state
<code>ConnectionAttributes</code>	Names of connection attributes
<code>ConnectionOperations</code>	Names of connection operations
<code>ConnectionNotification</code>	Constants and methods related to connection notifications
<code>DestinationAttributes</code>	Names of destination attributes
<code>DestinationOperations</code>	Names of destination operations
<code>DestinationNotification</code>	Constants and methods related to destination notifications
<code>DestinationType</code>	Names of destination types
<code>DestinationState</code>	Constants related to destination state

TABLE 2-2 Message Queue JMX Utility Classes (Continued)

Class	Description
DestinationLimitBehavior	Names of destination limit behaviors
DestinationPauseType	Constants related to destination pause type
ProducerAttributes	Names of message producer attributes
ProducerOperations	Names of message producer operations
ProducerInfo	Field names in composite data object for message producers
ConsumerAttributes	Names of message consumer attributes
ConsumerOperations	Names of message consumer operations
ConsumerInfo	Field names in composite data object for message consumers
TransactionAttributes	Names of transaction attributes
TransactionOperations	Names of transaction operations
TransactionNotification	Constants and methods related to transaction notifications
TransactionInfo	Field names in composite data object for transactions
TransactionState	Constants related to transaction state
ClusterAttributes	Names of broker cluster attributes
ClusterOperations	Names of broker cluster operations
ClusterNotification	Constants and methods related to broker cluster notifications
BrokerClusterInfo	Field names in composite data object for broker clusters
LogAttributes	Names of logging attributes
LogNotification	Constants and methods related to logging notifications
LogLevel	Names of logging levels
JVMAttributes	Names of Java Virtual Machine (JVM) attributes

Connecting to the MBean Server

As defined in the JMX Specification, client applications obtain MBeans through an *MBean server*, accessed by means of a *JMX connector*. Message Queue message brokers use the standard JMX-compliant MBean server and JMX connector provided with the Java Development Kit (JDK) 1.5, which use remote method invocation (RMI) as the infrastructure for communicating between client and server. Once you have a JMX connector, you can use it to obtain an *MBean server connection* with which to access the attributes, operations, and notifications of individual MBeans.

For convenience, Message Queue provides an *administration connection factory* (class `AdminConnectionFactory`), similar in spirit to the familiar Message Queue connection factory, for creating JMX connectors with a minimum of effort. It is also possible to dispense with this convenience class and obtain a JMX connector using standard JMX classes instead. The following sections illustrate these two techniques. While Message Queue client applications are free to use either method, the first is simpler and is recommended.

Obtaining a JMX Connector from an Administration Connection Factory

The Message Queue convenience class `AdminConnectionFactory` (defined in package `com.sun.messaging`) encapsulates a predefined set of configuration properties and hides the details involved in creating a JMX connector. [Example 2-1](#) shows the most straightforward use, creating a connector at the default port 7676 on host `localhost`, with the user name and password both set to the default value of `admin`. After creating the connector, its `getMBeanServerConnection` method is called to obtain a server connection for interacting with Message Queue MBeans.

EXAMPLE 2-1 Obtaining a JMX Connector from an Administration Connection Factory

```
import javax.management.remote.*;
import com.sun.messaging.AdminConnectionFactory;

// Create administration connection factory for default host and port (localhost:7676)
AdminConnectionFactory acf = new AdminConnectionFactory();

// Get JMX connector using default user name (admin) and password (admin)
JMXConnector jmx = acf.createConnection();

// Get MBean server connection
MBeanServerConnection mbsc = jmx.getMBeanServerConnection();
```

[Example 2-2](#) shows how to reconfigure an administration connection factory's properties to nondefault values. Instead of using the default broker address (`localhost:7676`), the code shown here uses the connection factory's `setProperty` method to reconfigure it to connect to a broker at port 9898 on host `otherhost`. (The names of the connection factory's configuration properties are defined as static constants in the Message Queue utility class `AdminConnectionFactory`, defined in package `com.sun.messaging`.) The arguments to the factory's `createConnection` method are then used to supply a user name and password other than the defaults.

EXAMPLE 2-2 Configuring an Administration Connection Factory

```
import javax.management.remote.*;
import com.sun.messaging.AdminConnectionFactory;

// Create administration connection factory
AdminConnectionFactory acf = new AdminConnectionFactory();

// Configure for specific broker address
acf.setProperty(AdminConnectionFactory.imqAddress, "otherhost:9898");

// Get JMX connector, supplying user name and password
JMXConnector jmx = acf.createConnection("AliBaba", "sesame");

// Get MBean server connection
MBeanServerConnection mbsc = jmx.getMBeanServerConnection();
```

Obtaining a JMX Connector Without Using an Administration Connection Factory

The generic (non-Message Queue) way of obtaining a JMX connector, as described in the JMX Specification, is by invoking the static `connect` method of the standard JMX class `JMXConnectorFactory` (see [Example 2-3](#)). Client applications may choose to use this method instead of an administration connection factory in order to avoid dependency on Message Queue-specific classes.

EXAMPLE 2-3 Obtaining a JMX Connector Without Using an Administration Connection Factory

```
import java.util.HashMap;
import javax.management.remote.*;

// Provide credentials required by server for user authentication
HashMap environment = new HashMap();
String[] credentials = new String[] {"AliBaba", "sesame"};
environment.put (JMXConnector.CREDENTIALS, credentials);

// Create JMXServiceURL of JMX Connector (must be known in advance)
JMXServiceURL url
    = new JMXServiceURL("service:jmx:rmi:///jndi/rmi://localhost:9999/server");

// Get JMX connector
JMXConnector jmx = JMXConnectorFactory.connect(url, environment);

// Get MBean server connection
```

EXAMPLE 2-3 Obtaining a JMX Connector Without Using an Administration Connection Factory
(Continued)

```
MBeanServerConnection mbsc = jmx.getMBeanServerConnection();
```

The `JMXConnectorFactory.connect` method accepts two parameters:

- A *JMX service URL* identifying the JMX connector server from which to obtain a connector
- An optional *environment* parameter specifying attributes of the connections to be made

The service URL is a string whose syntax is described in the next section; the environment parameter is a hash map mapping attribute names to their corresponding values. In particular, the `CREDENTIALS` attribute specifies the authentication credentials (user name and password) to be used in establishing a connection. The hash-map key for this attribute is defined as a static constant, `CREDENTIALS`, in the `JMXConnector` interface; the corresponding value is a 2-element string array containing the user name at index 0 and the password at index 1.

JMX Service URLs

For Message Queue applications (which always use the RMI protocol for JMX connections), the JMX service URL has the following syntax:

```
service:jmx:rmi://[host[:port]][urlPath]
```

Although *host* and *port* may be included, they are ignored by the RMI protocol. If *urlPath* is specified, it gives the Java Naming and Directory Interface (JNDI) location of an RMI stub (typically a location within an RMI registry) in the form

```
/jndi/jndiName
```

For example, the URL

```
service:jmx:rmi://myhost/jndi/rmi://myhost:1099/myhost/myjmxconnector
```

specifies an RMI stub at the location

```
rmi://myhost:1099/myhost/myjmxconnector
```

which is an RMI registry running at location `myhost/myjmxconnector` on port 1099 of host `myhost`.

Alternatively, if *urlPath* is omitted from the service URL, the JMX connector server will generate a client URL containing the actual RMI stub embedded within it in encoded and serialized form. For example, the service URL

```
service:jmx:rmi://localhost
```

will generate a client URL of the form

```
service:jmx:rmi://localhost/stub/rmiStub
```

where *rmiStub* is an encoded and serialized representation of the RMI stub itself.

Using MBeans

Once you have obtained an MBean server connection, you can use it to communicate with Message Queue (and other) MBeans and to access their attributes, operations, and notifications. The following sections describe how this is done.

Accessing MBean Attributes

The MBean server connection's `getAttribute` method accepts the object name of an MBean along with a string representing the name of one of its attributes, and returns the value of the designated attribute. [Example 2-4](#) shows an example, obtaining and printing the value of a destination's `MaxNumProducers` attribute from its configuration MBean (described in [“Destination Configuration” on page 66](#)).

EXAMPLE 2-4 Getting an Attribute Value

```
import javax.management.*;
import javax.management.remote.*;
import com.sun.messaging.AdminConnectionFactory;
import com.sun.messaging.jms.management.server.*;

public class GetAttrValue
{
    public static void main (String[] args)
    {
        try
        { // Create administration connection factory
            AdminConnectionFactory acf = new AdminConnectionFactory();

            // Get JMX connector, supplying user name and password
            JMXConnector jmxc = acf.createConnection("AliBaba", "sesame");

            // Get MBean server connection
            MBeanServerConnection mbsc = jmxc.getMBeanServerConnection();

            // Create object name
            ObjectName destConfigName
```

EXAMPLE 2-4 Getting an Attribute Value (Continued)

```

        = MQObjectName.createDestinationConfig(DestinationType.QUEUE, "MyQueue");

// Get and print attribute value
Integer attrValue
    = (Integer)mbsc.getAttribute(destConfigName, DestinationAttributes.MAX_NUM_PRODUCERS);
System.out.println( "Maximum number of producers: " + attrValue );

// Close JMX connector
jmxm.close();
}

catch (Exception e)
{ System.out.println( "Exception occurred: " + e.toString() );
  e.printStackTrace();
}
}
}
}

```

There is also an `MBeanServerConnection` method named `getAttributes`, which accepts an MBean object name and an array of attribute name strings, and returns a result of class `AttributeList`. This is an array of `Attribute` objects, each of which provides methods (`getName` and `getValue`) for retrieving the name and value of one of the requested attributes. [Example 2-5](#) shows a modified version of [Example 2-4](#) that uses `getAttributes` to retrieve the values of a destination's `MaxNumProducers` and `maxNumActiveConsumers` attributes from its configuration MBean (see [“Destination Configuration” on page 66](#)).

EXAMPLE 2-5 Getting Multiple Attribute Values

```

import javax.management.*;
import javax.management.remote.*;
import com.sun.messaging.AdminConnectionFactory;
import com.sun.messaging.jms.management.server.*;

public class GetAttrValues
{
    public static void main (String[] args)
    {
        try
        { // Create administration connection factory
          AdminConnectionFactory acf = new AdminConnectionFactory();

          // Get JMX connector, supplying user name and password
          JMXConnector jmxm = acf.createConnection("AliBaba", "sesame");

```

EXAMPLE 2-5 Getting Multiple Attribute Values (Continued)

```

// Get MBean server connection
MBeanServerConnection mbsc = jmxc.getMBeanServerConnection();

// Create object name
ObjectName destConfigName
    = MQObjectName.createDestinationConfig(DestinationType.QUEUE, "MyQueue");

// Create array of attribute names
String attrNames[] =
    { DestinationAttributes.MAX_NUM_PRODUCERS,
      DestinationAttributes.MAX_NUM_ACTIVE_CONSUMERS
    };

// Get attributes
AttributeList attrList = mbsc.getAttributes(destConfigName, attrNames);

// Extract and print attribute values

Object attrValue;

attrValue = attrList.get(0).getValue();
System.out.println( "Maximum number of producers: " + attrValue.toString() );

attrValue = attrList.get(1).getValue();
System.out.println( "Maximum number of active consumers: " + attrValue.toString() );

// Close JMX connector
jmxc.close();
}

catch (Exception e)
{ System.out.println( "Exception occurred: " + e.toString() );
  e.printStackTrace();
}
}
}

```

To set the value of an attribute, use the `MBeanServerConnection` method `setAttribute`. This takes an MBean object name and an `Attribute` object specifying the name and value of the attribute to be set. [Example 2-6](#) uses this method to set a destination's `MaxNumProducers` attribute to 25.

EXAMPLE 2-6 Setting an Attribute Value

```

import javax.management.*;
import javax.management.remote.*;

```

EXAMPLE 2-6 Setting an Attribute Value (Continued)

```
import com.sun.messaging.AdminConnectionFactory;
import com.sun.messaging.jms.management.server.*;

public class SetAttrValue
{
    public static void main (String[] args)
    {
        try
        { // Create administration connection factory
            AdminConnectionFactory acf = new AdminConnectionFactory();

            // Get JMX connector, supplying user name and password
            JMXConnector jmx = acf.createConnection("AliBaba", "sesame");

            // Get MBean server connection
            MBeanServerConnection mbsc = jmx.getMBeanServerConnection();

            // Create object name
            ObjectName destConfigName
                = MQObjectName.createDestinationConfig(DestinationType.QUEUE, "MyQueue");

            // Create attribute object
            Attribute attr = new Attribute(DestinationAttributes.MAX_NUM_PRODUCERS, 25);

            // Set attribute value
            mbsc.setAttribute(destConfigName, attr);

            // Close JMX connector
            jmx.close();
        }

        catch (Exception e)
        { System.out.println( "Exception occurred: " + e.toString() );
          e.printStackTrace();
        }
    }
}
```

Just as for getting attribute values, there is an `MBeanServerConnection` method named `setAttributeValues` for setting the values of multiple attributes at once. You supply an MBean object name and an attribute list giving the names and values of the attributes to be set. [Example 2-7](#) illustrates the use of this method to set a destination's `MaxNumProducers` and `MaxNumActiveConsumers` attributes to 25 and 50, respectively.

EXAMPLE 2-7 Setting Multiple Attribute Values

```
import javax.management.*;
import javax.management.remote.*;
import com.sun.messaging.AdminConnectionFactory;
import com.sun.messaging.jms.management.server.*;

public class SetAttrValues
{
    public static void main (String[] args)
    {
        try
        {
            // Create administration connection factory
            AdminConnectionFactory acf = new AdminConnectionFactory();

            // Get JMX connector, supplying user name and password
            JMXConnector jmxc = acf.createConnection("AliBaba", "sesame");

            // Get MBean server connection
            MBeanServerConnection mbsc = jmxc.getMBeanServerConnection();

            // Create object name
            ObjectName destConfigName
                = MQObjectName.createDestinationConfig(DestinationType.QUEUE, "MyQueue");

            // Create and populate attribute list

            AttributeList attrList = new AttributeList();
            Attribute attr;

            attr = new Attribute(DestinationAttributes.MAX_NUM_PRODUCERS, 25);
            attrList.add(attr);

            attr = new Attribute(DestinationAttributes.MAX_NUM_ACTIVE_CONSUMERS, 50);
            attrList.add(attr);

            // Set attribute values
            mbsc.setAttributes(destConfigName, attrList);

            // Close JMX connector
            jmxc.close();
        }
        catch (Exception e)
        {
            System.out.println("Exception occurred: " + e.toString());
            e.printStackTrace();
        }
    }
}
```

EXAMPLE 2-7 Setting Multiple Attribute Values (Continued)

```
    }
}
```

Invoking MBean Operations

To invoke an MBean operation, use the `MBeanServerConnection` method `invoke`. The first two parameters to this method are an MBean object name and a string specifying the name of the operation to be invoked. (The two remaining parameters are used for supplying parameters to the invoked operation, and are discussed in the next example.) The method returns an object that is the operation's return value (if any). [Example 2-8](#) shows the use of this method to pause the `.jms` connection service by invoking the `pause` operation of its service configuration MBean (see “[Service Configuration](#)” on page 53).

EXAMPLE 2-8 Invoking an Operation

```
import javax.management.*;
import javax.management.remote.*;
import com.sun.messaging.AdminConnectionFactory;
import com.sun.messaging.jms.management.server.*;

public class InvokeOp
{
    public static void main (String[] args)
    {
        try
        { // Create administration connection factory
            AdminConnectionFactory acf = new AdminConnectionFactory();

            // Get JMX connector, supplying user name and password
            JMXConnector jmxc = acf.createConnection("AliBaba", "sesame");

            // Get MBean server connection
            MBeanServerConnection mbsc = jmxc.getMBeanServerConnection();

            // Create object name
            ObjectName serviceConfigName = MQObjectName.createServiceConfig("jms");

            // Invoke operation
            mbsc.invoke(serviceConfigName, ServiceOperations.PAUSE, null, null);

            // Close JMX connector
            jmxc.close();
        }
    }
}
```

EXAMPLE 2-8 Invoking an Operation (Continued)

```

    catch (Exception e)
    { System.out.println( "Exception occurred: " + e.toString() );
      e.printStackTrace();
    }
  }
}

```

When the operation being invoked requires parameters, you supply them in an array as the third parameter to the `MBeanServerConnection.invoke` method. The method's fourth parameter is a signature array giving the class or interface names of the invoked operation's parameters. [Example 2-9](#) shows an illustration, invoking the destination manager configuration MBean's `create` operation to create a new queue destination named `MyQueue` with the same attributes that were set in [Example 2-7](#). The `create` operation (see [“Destination Manager Configuration” on page 75](#)) takes three parameters: the type (`QUEUE` or `TOPIC`) and name of the new destination and an attribute list specifying any initial attribute values to be set. The example shows how to set up a parameter array (`opParams`) containing these values, along with a signature array (`opSig`) giving their classes, and pass them to the `invoke` method.

EXAMPLE 2-9 Invoking an Operation with Parameters

```

import javax.management.*;
import javax.management.remote.*;
import com.sun.messaging.AdminConnectionFactory;
import com.sun.messaging.jms.management.server.*;

public class InvokeOpWithParams
{
    public static void main (String[] args)
    {
        try
        { // Create administration connection factory
          AdminConnectionFactory acf = new AdminConnectionFactory();

          // Get JMX connector, supplying user name and password
          JMXConnector jmx = acf.createConnection("AliBaba", "sesame");

          // Get MBean server connection
          MBeanServerConnection mbsc = jmx.getMBeanServerConnection();

          // Create object name
          ObjectName destMgrConfigName
            = new ObjectName(MQObjectName.DESTINATION_MANAGER_CONFIG_MBEAN_NAME);

```

EXAMPLE 2-9 Invoking an Operation with Parameters (Continued)

```

// Create and populate attribute list

AttributeList attrList = new AttributeList();
Attribute      attr;

attr = new Attribute(DestinationAttributes.MAX_NUM_PRODUCERS, 25);
attrList.add(attr);

attr = new Attribute(DestinationAttributes.MAX_NUM_ACTIVE_CONSUMERS, 50);
attrList.add(attr);

// Create operation's parameter and signature arrays

Object  opParams[] = { DestinationType.QUEUE,
                      "MyQueue",
                      attrList
                    };

String  opSig[] = { String.class.getName(),
                  String.class.getName(),
                  attrList.getClass().getName()
                };

// Invoke operation
mbsc.invoke(destMgrConfigName, DestinationOperations.CREATE, opParams, opSig);

// Close JMX connector
jmxm.close();
}

catch (Exception e)
{ System.out.println( "Exception occurred: " + e.toString() );
  e.printStackTrace();
}
}
}

```

Example 2-10 shows a more elaborate example combining the use of MBean operations and attributes. The destination manager monitor MBean operation `getDestinations` (see “[Destination Manager Monitor](#)” on page 78) returns an array of object names of the destination monitor MBeans for all current destinations. The example then iterates through the array, printing the name, destination type (QUEUE or TOPIC), and current state (such as RUNNING or PAUSED) for each destination.

EXAMPLE 2-10 Combining Operations and Attributes

```
import javax.management.*;
import javax.management.remote.*;
import com.sun.messaging.AdminConnectionFactory;
import com.sun.messaging.jms.management.server.*;

public class OpsAndAttrs
{
    public static void main (String[] args)
    {
        try
        { // Create administration connection factory
            AdminConnectionFactory acf = new AdminConnectionFactory();

            // Get JMX connector, supplying user name and password
            JMXConnector jmxc = acf.createConnection("AliBaba", "sesame");

            // Get MBean server connection
            MBeanServerConnection mbsc = jmxc.getMBeanServerConnection();

            // Create object name for destination manager monitor MBean
            ObjectName destMgrMonitorName
                = new ObjectName(MQObjectName.DESTINATION_MANAGER_MONITOR_MBEAN_NAME);

            // Get destination object names
            ObjectName destNames[] = mbsc.invoke(destMgrMonitorName,
                DestinationOperations.GET_DESTINATIONS,
                null,
                null);

            // Step through array of object names, printing information for each destination

            System.out.println( "Listing destinations: " );

            ObjectName eachDestName;
            Object attrValue;

            for ( int i = 0; i < destNames.length; ++i )
            { eachDestName = destNames[i];

                attrValue = mbsc.getAttribute(eachDestName, DestinationAttributes.NAME);
                System.out.println( "\tName: " + attrValue );

                attrValue = mbsc.getAttribute(eachDestName, DestinationAttributes.TYPE);
                System.out.println( "\tTypeYPE: " + attrValue );
            }
        }
    }
}
```

EXAMPLE 2-10 Combining Operations and Attributes *(Continued)*

```

        attrValue = mbsc.getAttribute(eachDestName, DestinationAttributes.STATE_LABEL);
        System.out.println( "\tState: " + attrValue );

        System.out.println( "" );
    }

    // Close JMX connector
    jmx.c.close();
}

catch (Exception e)
{ System.out.println( "Exception occurred: " + e.toString() );
  e.printStackTrace();
}
}
}
}

```

Some of the Message Queue MBeans' operations and attributes return a *composite data* object (implementing the JMX `CompositeData` interface). This type of object consists of a collection of data values accessed by means of associative *lookup keys*. The specific keys vary from one MBean to another, and are described in the relevant sections of [Chapter 3, "Message Queue MBean Reference."](#) [Example 2-11](#) shows an illustration, invoking the consumer manager MBean's `GetConsumerInfo` operation (see ["Consumer Manager Monitor" on page 85](#) to obtain an array of composite data objects describing all current message consumers. It then steps through the array, using the lookup keys listed in [Table 3-63](#) to retrieve and print the characteristics of each consumer.

EXAMPLE 2-11 Using a Composite Data Object

```

import javax.management.*;
import javax.management.remote.*;
import com.sun.messaging.AdminConnectionFactory;
import com.sun.messaging.jms.management.server.*;

public class CompData
{
    public static void main (String[] args)
    {
        try
        { // Create administration connection factory
          AdminConnectionFactory acf = new AdminConnectionFactory();

          // Get JMX connector, supplying user name and password
          JMXConnector jmx.c = acf.createConnection("AliBaba", "sesame");

```

EXAMPLE 2-11 Using a Composite Data Object (Continued)

```
// Get MBean server connection
MBeanServerConnection mbsc = jmx.getMBeanServerConnection();

// Create object name
ObjectName consumerMgrMonitorName
    = new ObjectName(MQObjectName.CONSUMER_MANAGER_MONITOR_MBEAN_NAME);

// Invoke operation
Object result
    = mbsc.invoke(consumerMgrMonitorName, ConsumerOperations.GET_CONSUMER_INFO, null, null);

// Typecast result to an array of composite data objects
CompositeData cdArray[] = (CompositeData[])result;

// Step through array, printing information for each consumer

if ( cdArray == null )
    { System.out.println( "No message consumers found" );
    }
else
    { for ( int i = 0; i < cdArray.length; ++i )
        { CompositeData cd = cdArray[i];

            System.out.println( "Consumer ID: "
                + cd.get(ConsumerInfo.CONSUMER_ID) );
            System.out.println( "User: "
                + cd.get(ConsumerInfo.USER) );
            System.out.println( "Host: "
                + cd.get(ConsumerInfo.HOST) );
            System.out.println( "Connection service: "
                + cd.get(ConsumerInfo.SERVICE_NAME) );
            System.out.println( "Acknowledgment mode: "
                + cd.get(ConsumerInfo.ACKNOWLEDGE_MODE_LABEL) );
            System.out.println( "Destination name: "
                + cd.get(ConsumerInfo.DESTINATION_NAME) );
            System.out.println( "Destination type: "
                + cd.get(ConsumerInfo.DESTINATION_TYPE) );

        }
    }

}

catch (Exception e)
    { System.out.println( "Exception occurred: " + e.toString() );
      e.printStackTrace();
    }
}
```

EXAMPLE 2-11 Using a Composite Data Object (Continued)

```

finally
{ if ( jmxrc != null )
  { try
    { jmxrc.close();
    }
    catch (IOException ioe)
    { System.out.println( "I/O exception occurred: " + ioe.toString() );
      ioe.printStackTrace();
    }
  }
}
}
}
}

```

Receiving MBean Notifications

To receive notifications from an MBean, you must register a *notification listener* with the MBean server. This is an object implementing the JMX interface `NotificationListener`, which consists of the single method `handleNotification`. In registering the listener with the MBean server (using the `MBeanServerConnection` method `addNotificationListener`), you supply the object name of the MBean from which you wish to receive notifications, along with a *notification filter* specifying which types of notification you wish to receive. (You can also provide an optional *handback object* to be passed to your listener whenever it is invoked, and which you can use for any purpose convenient to your application.) The MBean server will then call your listener's `handleNotification` method whenever the designated MBean broadcasts a notification satisfying the filter you specified.

The notification listener's `handleNotification` method receives two parameters: a *notification object* (belonging to the JMX class `Notification`) describing the notification being raised, along with the handback object, if any, that you supplied when you registered the listener. The notification object provides methods for retrieving various pieces of information about the notification, such as its type, the MBean raising it, its time stamp, and an MBean-dependent *user data* object and *message string* further describing the notification. The notifications raised by Message Queue MBeans belong to Message Queue–specific subclasses of `Notification`, such as `BrokerNotification`, `ServiceNotification`, and `DestinationNotification`, which add further information retrieval methods specific to each particular type of notification; see the relevant sections of [Chapter 3, “Message Queue MBean Reference”](#) for details.

[Example 2-12](#) shows a notification listener for responding to Message Queue service notifications, issued by a service manager monitor MBean. On receiving a notification belonging to the Message Queue class `ServiceNotification`, the listener simply prints an informational message containing the notification's type and the name of the connection service affected.

EXAMPLE 2-12 Notification Listener

```

import javax.management.*;
import javax.management.remote.*;
import com.sun.messaging.jms.management.server.*;

public class ServiceNotificationListener implements NotificationListener
{
    public void handleNotification (Notification notification,
                                   Object          handback)
    {
        if ( notification instanceof ServiceNotification )
            { ServiceNotification n = (ServiceNotification)notification;
              }
        else
            { System.err.println( "Wrong type of notification for listener" );
              return;
            }

        System.out.println( "\nReceived service notification: " );
        System.out.println( "\tNotification type: " + n.getType() );
        System.out.println( "\tService name: " + n.getServiceName() );

        System.out.println( "" );
    }
}

```

[Example 2-13](#) shows how to register the notification listener from [Example 2-12](#), using the `MBeanServerConnection` method `addNotificationListener`. The notification filter is an object of the standard JMX class `NotificationFilterSupport`; the calls to this object's `enableType` method specify that the listener should be invoked whenever a connection service is paused or resumed. The listener itself is an instance of class `ServiceNotificationListener`, as defined in [Example 2-12](#).

EXAMPLE 2-13 Registering a Notification Listener

```

import javax.management.*;
import javax.management.remote.*;
import com.sun.messaging.AdminConnectionFactory;
import com.sun.messaging.jms.management.server.*;
import java.io.IOException;

public class NotificationService
{
    public static void main (String[] args)

```

EXAMPLE 2-13 Registering a Notification Listener (Continued)

```
{
  try
  { // Create administration connection factory
    AdminConnectionFactory acf = new AdminConnectionFactory();

    // Get JMX connector, supplying user name and password
    JMXConnector jmxcr = acf.createConnection("AliBaba", "sesame");

    // Get MBean server connection
    MBeanServerConnection mbsc = jmxcr.getMBeanServerConnection();

    // Create object name for service manager monitor MBean
    ObjectName svcMgrMonitorName
      = new ObjectName( MQObjectName.SERVICE_MANAGER_MONITOR_MBEAN_NAME );

    // Create notification filter
    NotificationFilterSupport myFilter = new NotificationFilterSupport();
    myFilter.enableType(ServiceNotification.SERVICE_PAUSE);
    myFilter.enableType(ServiceNotification.SERVICE_RESUME);

    // Create notification listener
    ServiceNotificationListener myListener = new ServiceNotificationListener();
    mbsc.addNotificationListener(svcMgrMonitorName, myListener, myFilter, null);

    ...
  }

  catch (Exception e)
  { System.out.println( "Exception occurred: " + e.toString() );
    e.printStackTrace();
  }

  finally
  { if ( jmxcr != null )
    { try
      { jmxcr.close();
        }
      catch (IOException ioe)
      { System.out.println( "I/O exception occurred: " + ioe.toString() );
        ioe.printStackTrace();
      }
    }
  }
}
```

Message Queue MBean Reference

This chapter describes the JMX MBeans that allow you to configure and monitor a Message Queue broker. It consists of the following sections:

- “Message Brokers” on page 47
- “Connection Services” on page 53
- “Connections” on page 61
- “Destinations” on page 66
- “Message Producers” on page 80
- “Message Consumers” on page 83
- “Transactions” on page 88
- “Broker Clusters” on page 92
- “Logging” on page 100
- “Java Virtual Machine” on page 103

Message Brokers

This section describes the MBeans used for managing message brokers:

- The broker configuration MBean configures a message broker.
- The broker monitor MBean monitors a message broker.

The following subsections describe each of these MBeans in detail.

Broker Configuration

The *broker configuration MBean* is used for configuring a message broker. There is one such MBean for each broker.

Object Name

The broker configuration MBean has the following object name:

```
com.sun.messaging.jms.server:type=Broker,subtype=Config
```

A string representing this object name is defined as a static constant `BROKER_CONFIG_MBEAN_NAME` in the utility class `MQObjectName`.

Attributes

The broker configuration MBean has the attributes shown in [Table 3–1](#). The names of these attributes are defined as static constants in the utility class `BrokerAttributes`.

TABLE 3–1 Broker Configuration Attributes

Name	Type	Settable?	Description
BrokerID	String	No	<p>Broker identifier</p> <p>Must be unique; no two running brokers may have the same broker identifier.</p> <p>For brokers using a JDBC-based persistent data store, this string is appended to the names of all database tables to make them unique when more than one broker instance is using the same database. Must be an alphanumeric string of no more than $n - 13$ characters, where n is the maximum table name length allowed by the database. If a database is not used as the persistent data store, the value of this attribute is <code>null</code>.</p> <p>Note – For high-availability brokers, database table names use the <code>ClusterID</code> attribute (see Table 3–74) instead.</p>
Version	String	No	Broker version
InstanceName	String	No	<p>Broker instance name</p> <p>Example: <code>imqbroker</code></p>
Port	Integer	Yes	Port number of Port Mapper

Operations

The broker configuration MBean supports the operations shown in [Table 3–2](#). The names of these operations are defined as static constants in the utility class `BrokerOperations`.

TABLE 3-2 Broker Configuration Operations

Name	Parameters	Result Type	Description
shutdown	<i>nofailover</i> (Boolean) <i>time</i> (Long)	None	Shut down broker If <i>nofailover</i> is <code>false</code> or <code>null</code> , another broker will attempt to take over for this broker when it shuts down; this applies only to brokers in a high-availability (HA) cluster. If <i>nofailover</i> is <code>true</code> , no such takeover attempt will occur. The <i>time</i> parameter specifies the interval, in seconds, before the broker actually shuts down; for immediate shutdown, specify <code>0</code> or <code>null</code> .
shutdown	None	None	Shut down broker immediately If the broker is part of a high-availability (HA) cluster, another broker will attempt to take over for it. Equivalent to <code>shutdown(Boolean.FALSE, new Long(0))</code> .
restart	None	None	Restart broker
quiesce	None	None	Quiesce broker The broker will refuse any new connections; existing connections will continue to be served.
unquiesce	None	None	Unquiesce broker The broker will again accept new connections.
takeover ¹	<i>brokerID</i> (String)	None	Initiate takeover from specified broker The desired broker is designated by its broker identifier (<i>brokerID</i>).
resetMetrics	None	None	Reset metrics Resets to zero all metrics in monitor MBeans that track cumulative, peak, or average counts. The following attributes are affected: Service monitor <ul style="list-style-type: none"> NumConnectionsOpened NumConnectionsRejected NumMsgsIn NumMsgsOut MsgBytesIn MsgBytesOut NumPktsIn NumPktsOut PktBytesIn PktBytesOut

¹ HA clusters only

TABLE 3-2 Broker Configuration Operations (Continued)

Name	Parameters	Result Type	Description
			<p>Service manager monitor</p> <ul style="list-style-type: none"> NumMsgsIn NumMsgsOut MsgBytesIn MsgBytesOut NumPktsIn NumPktsOut PktBytesIn PktBytesOut <p>Connection manager monitor</p> <ul style="list-style-type: none"> NumConnectionsOpened NumConnectionsRejected <p>Destination monitor</p> <ul style="list-style-type: none"> PeakNumConsumers AvgNumConsumers PeakNumActiveConsumers AvgNumActiveConsumers PeakNumBackupConsumers AvgNumBackupConsumers PeakNumMsgs AvgNumMsgs NumMsgsIn NumMsgsOut MsgBytesIn MsgBytesOut PeakMsgBytes PeakTotalMsgBytes AvgTotalMsgBytes <p>Transaction manager monitor</p> <ul style="list-style-type: none"> NumTransactionsCommitted NumTransactionsRollback

Notification

The broker configuration MBean supports the notification shown in [Table 3-3](#).

TABLE 3-3 Broker Configuration Notification

Name	Description
<code>jmx.attribute.change</code>	Attribute value changed

Broker Monitor

The *broker monitor MBean* is used for monitoring a message broker. There is one such MBean for each broker.

Object Name

The broker monitor MBean has the following object name:

```
com.sun.messaging.jms.server:type=Broker,subtype=Monitor
```

A string representing this object name is defined as a static constant `BROKER_MONITOR_MBEAN_NAME` in the utility class `MQObjectName`.

Attributes

The broker monitor MBean has the attributes shown in [Table 3-4](#). The names of these attributes are defined as static constants in the utility class `BrokerAttributes`.

TABLE 3-4 Broker Monitor Attributes

Name	Type	Settable?	Description
<code>BrokerID</code>	String	No	<p>Broker identifier</p> <p>Must be unique; no two running brokers may have the same broker identifier.</p> <p>For brokers using a JDBC-based persistent data store, this string is appended to the names of all database tables to make them unique when more than one broker instance is using the same database. Must be an alphanumeric string of no more than $n - 13$ characters, where n is the maximum table name length allowed by the database. If a database is not used as the persistent data store, the value of this attribute is <code>null</code>.</p> <p>Note – For high-availability brokers, database table names use the <code>ClusterID</code> attribute (see Table 3-78) instead.</p>
<code>Version</code>	String	No	Broker version
<code>InstanceName</code>	String	No	Broker instance name
<code>Port</code>	Integer	No	Port number of Port Mapper

TABLE 3-4 Broker Monitor Attributes (Continued)

Name	Type	Settable?	Description
Embedded	Boolean	No	Is broker embedded (started from within another process)?

Notifications

The broker monitor MBean supports the notifications shown in Table 3-5. These notifications are instances of the Message Queue JMX classes `BrokerNotification` and `ClusterNotification`, and their names are defined as static constants in those classes.

TABLE 3-5 Broker Monitor Notifications

Name	Utility Constant	Description
<code>mq.broker.shutdown.start</code>	<code>BrokerNotification.BROKER_SHUTDOWN_START</code>	Broker has begun shutting down
<code>mq.broker.quiesce.start</code>	<code>BrokerNotification.BROKER QUIESCE_START</code>	Broker has begun quiescing
<code>mq.broker.quiesce.complete</code>	<code>BrokerNotification.BROKER QUIESCE_COMPLETE</code>	Broker has finished quiescing
<code>mq.broker.takeover.start</code> ¹	<code>BrokerNotification.BROKER TAKEOVER_START</code>	Broker has begun taking over persistent data store from another broker
<code>mq.broker.takeover.complete</code> ¹	<code>BrokerNotification.BROKER TAKEOVER_COMPLETE</code>	Broker has finished taking over persistent data store from another broker
<code>mq.broker.takeover.fail</code> ¹	<code>BrokerNotification.BROKER TAKEOVER_FAIL</code>	Attempted takeover has failed
<code>mq.cluster.broker.join</code>	<code>ClusterNotification.CLUSTER_BROKER_JOIN</code>	Broker has joined a cluster

¹ HA clusters only

Table 3-6 shows the methods defined in class `BrokerNotification` for obtaining details about a broker monitor notification. See Table 3-83 for the corresponding methods of class `ClusterNotification`.

TABLE 3-6 Data Retrieval Methods for Broker Monitor Notifications

Method	Result Type	Description
<code>getBrokerID</code>	String	Broker identifier
<code>getBrokerAddress</code>	String	Broker address, in the form <code>hostName:portNumber</code> Example: <code>host1:3000</code>

TABLE 3-6 Data Retrieval Methods for Broker Monitor Notifications (Continued)

Method	Result Type	Description
getFailedBrokerID ¹	String	Broker identifier of broker being taken over

¹ HA clusters only

Connection Services

This section describes the MBeans used for managing connection services:

- The service configuration MBean configures a connection service.
- The service monitor MBean monitors a connection service.
- The service manager configuration MBean manages service configuration MBeans.
- The service manager monitor MBean manages service monitor MBeans.

The following subsections describe each of these MBeans in detail.

Service Configuration

The *service configuration MBean* is used for configuring a connection service. There is one such MBean for each service.

Object Name

The service configuration MBean has an object name of the following form:

```
com.sun.messaging.jms.server:type=Service,subtype=Config,name=serviceName
```

where *serviceName* is the name of the connection service (see [Table 3-7](#)). The utility class `MQObjectName` provides a static method, `createServiceConfig`, for constructing object names of this form.

TABLE 3-7 Connection Service Names for Service Configuration MBeans

Service Name	Service Type	Protocol Type
jms	Normal	TCP
ssljms	Normal	TLS (SSL-based security)
httpjms	Normal	HTTP
httpsjms	Normal	HTTPS (SSL-based security)
admin	Admin	TCP

TABLE 3-7 Connection Service Names for Service Configuration MBeans *(Continued)*

Service Name	Service Type	Protocol Type
ssladmin	Admin	TLS (SSL-based security)

Attributes

The service configuration MBean has the attributes shown in [Table 3-8](#). The names of these attributes are defined as static constants in the utility class `ServiceAttributes`.

TABLE 3-8 Service Configuration Attributes

Name	Type	Settable?	Description
Name	String	No	Service name See Table 3-7 for possible values.
Port	Integer	Yes	Port number (<code>jms</code> , <code>ssljms</code> , <code>admin</code> , and <code>ssladmin</code> services only) A value of 0 specifies that the port is to be dynamically allocated by the Port Mapper; to learn the actual port currently used by the service, use the <code>Port</code> attribute of the service monitor MBean.
MinThreads	Integer	Yes	Minimum number of threads assigned to service Must be greater than 0.
MaxThreads	Integer	Yes	Maximum number of threads assigned to service Must be greater than or equal to <code>MinThreads</code> .
ThreadPoolModel	String	No	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 (<code>jms</code> and <code>admin</code> services only)

Operations

The service configuration MBean supports the operations shown in [Table 3-9](#). The names of these operations are defined as static constants in the utility class `ServiceOperations`.

TABLE 3-9 Service Configuration Operations

Name	Parameters	Result Type	Description
pause	None	None	Pause service (<code>jms</code> , <code>ssljms</code> , <code>httpjms</code> , and <code>httpsjms</code> services only)
resume	None	None	Resume service (<code>jms</code> , <code>ssljms</code> , <code>httpjms</code> , and <code>httpsjms</code> services only)

Notification

The service configuration MBean supports the notification shown in [Table 3–10](#).

TABLE 3–10 Service Configuration Notification

Name	Description
<code>jmx.attribute.change</code>	Attribute value changed

Service Monitor

The *service monitor MBean* is used for monitoring a connection service. There is one such MBean for each service.

Object Name

The service monitor MBean has an object name of the following form:

```
com.sun.messaging.jms.server:type=Service,subtype=Monitor,name=serviceName
```

where *serviceName* is the name of the connection service (see [Table 3–11](#)). The utility class `MQObjectName` provides a static method, `createServiceMonitor`, for constructing object names of this form.

TABLE 3–11 Connection Service Names for Service Monitor MBeans

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

Attributes

The service monitor MBean has the attributes shown in [Table 3–12](#). The names of these attributes are defined as static constants in the utility class `ServiceAttributes`.

TABLE 3-12 Service Monitor Attributes

Name	Type	Settable?	Description
Name	String	No	Service name See Table 3-11 for possible values.
Port	Integer	No	Port number currently used by service
State	Integer	No	Current state See Table 3-13 for possible values.
StateLabel	String	No	String representation of current state: Useful for displaying the state in human-readable form, such as in the Java Monitoring and Management Console (jconsole). See Table 3-13 for possible values.
NumConnections	Integer	No	Current number of connections
NumConnectionsOpened	Long	No	Cumulative number of connections opened since broker started
NumConnectionsRejected	Long	No	Cumulative number of connections rejected since broker started
NumActiveThreads	Integer	No	Current number of threads actively handling connections
NumProducers	Integer	No	Current number of message producers
NumConsumers	Integer	No	Current number of message consumers
NumMsgsIn	Long	No	Cumulative number of messages received since broker started
NumMsgsOut	Long	No	Cumulative number of messages sent since broker started
MsgBytesIn	Long	No	Cumulative size in bytes of messages received since broker started
MsgBytesOut	Long	No	Cumulative size in bytes of messages sent since broker started
NumPktsIn	Long	No	Cumulative number of packets received since broker started
NumPktsOut	Long	No	Cumulative number of packets sent since broker started
PktBytesIn	Long	No	Cumulative size in bytes of packets received since broker started
PktBytesOut	Long	No	Cumulative size in bytes of packets sent since broker started

[Table 3-13](#) shows the possible values for the `State` and `StateLabel` attributes. These values are defined as static constants in the utility class `ServiceState`.

TABLE 3-13 Connection Service State Values

Value	Utility Constant	String Representation	Meaning
0	ServiceState.RUNNING	RUNNING	Service running
1	ServiceState.PAUSED	PAUSED	Service paused
2	ServiceState.QUIESCED	QUIESCED	Service quiesced
-1	ServiceState.UNKNOWN	UNKNOWN	Service state unknown

Operations

The service monitor MBean supports the operations shown in [Table 3-14](#). The names of these operations are defined as static constants in the utility class `ServiceOperations`.

TABLE 3-14 Service Monitor Operations

Name	Parameters	Result Type	Description
<code>getConnections</code>	None	<code>ObjectName[]</code>	Object names of connection monitor MBeans for all current connections
<code>getProducerIDs</code>	None	<code>String[]</code>	Producer identifiers of all current message producers
<code>getConsumerIDs</code>	None	<code>String[]</code>	Consumer identifiers of all current message consumers

Notifications

The service monitor MBean supports the notifications shown in [Table 3-15](#). These notifications are instances of the Message Queue JMX classes `ServiceNotification` and `ConnectionNotification`, and their names are defined as static constants in those classes.

TABLE 3-15 Service Monitor Notifications

Name	Utility Constant	Description
<code>mq.service.pause</code>	<code>ServiceNotification.SERVICE_PAUSE</code>	Service paused
<code>mq.service.resume</code>	<code>ServiceNotification.SERVICE_RESUME</code>	Service resumed
<code>mq.connection.open</code>	<code>ConnectionNotification.CONNECTION_OPEN</code>	Connection opened
<code>mq.connection.reject</code>	<code>ConnectionNotification.CONNECTION_REJECT</code>	Connection rejected
<code>mq.connection.close</code>	<code>ConnectionNotification.CONNECTION_CLOSE</code>	Connection closed

[Table 3–16](#) shows the method defined in class `ServiceNotification` for obtaining details about a service monitor notification. See [Table 3–31](#) for the corresponding methods of class `ConnectionNotification`.

TABLE 3–16 Data Retrieval Method for Service Monitor Notifications

Method	Result Type	Description
<code>getServiceName</code>	String	Service name See Table 3–11 for possible values.

Service Manager Configuration

Each broker has a single *service manager configuration MBean*, used for managing all of the broker's service configuration MBeans.

Object Name

The service manager configuration MBean has the following object name:

```
com.sun.messaging.jms.server:type=ServiceManager,subtype=Config
```

A string representing this object name is defined as a static constant `SERVICE_MANAGER_CONFIG_MBEAN_NAME` in the utility class `MQObjectName`.

Attributes

The service manager configuration MBean has the attributes shown in [Table 3–17](#). The names of these attributes are defined as static constants in the utility class `ServiceAttributes`.

TABLE 3–17 Service Manager Configuration Attributes

Name	Type	Settable?	Description
<code>MinThreads</code>	Integer	No	Total minimum number of threads for all active services
<code>MaxThreads</code>	Integer	No	Total maximum number of threads for all active services

Operations

The service manager configuration MBean supports the operations shown in [Table 3–18](#). The names of these operations are defined as static constants in the utility class `ServiceOperations`.

TABLE 3-18 Service Manager Configuration Operations

Name	Parameters	Result Type	Description
getServices	None	ObjectName[]	Object names of service configuration MBeans for all services
pause	None	None	Pause all services except admin and ssladmin
resume	None	None	Resume all services

Service Manager Monitor

Each broker has a single *service manager monitor MBean*, used for managing all of the broker's service monitor MBeans.

Object Name

The service manager monitor MBean has the following object name:

```
com.sun.messaging.jms.server:type=ServiceManager,subtype=Monitor
```

A string representing this object name is defined as a static constant `SERVICE_MANAGER_MONITOR_MBEAN_NAME` in the utility class `MQObjectName`.

Attributes

The service manager monitor MBean has the attributes shown in [Table 3-19](#). The names of these attributes are defined as static constants in the utility class `ServiceAttributes`.

TABLE 3-19 Service Manager Monitor Attributes

Name	Type	Settable?	Description
NumServices	Integer	No	Number of connection services
NumActiveThreads	Integer	No	Total current number of threads actively handling connections for all services
NumMsgsIn	Long	No	Total cumulative number of messages received by all services since broker started
NumMsgsOut	Long	No	Total cumulative number of messages sent by all services since broker started
MsgBytesIn	Long	No	Total cumulative size in bytes of messages received by all services since broker started
MsgBytesOut	Long	No	Total cumulative size in bytes of messages sent by all services since broker started

TABLE 3–19 Service Manager Monitor Attributes (Continued)

Name	Type	Settable?	Description
NumPktsIn	Long	No	Total cumulative number of packets received by all services since broker started
NumPktsOut	Long	No	Total cumulative number of packets sent by all services since broker started
PktBytesIn	Long	No	Total cumulative size in bytes of packets received by all services since broker started
PktBytesOut	Long	No	Total cumulative size in bytes of packets sent by all services since broker started

Operation

The service manager monitor MBean supports the operation shown in [Table 3–20](#). The name of this operation is defined as a static constant in the utility class `ServiceOperations`.

TABLE 3–20 Service Manager Monitor Operation

Name	Parameters	Result Type	Description
getServices	None	ObjectName[]	Object names of all service monitor MBeans

Notifications

The service manager monitor MBean supports the notifications shown in [Table 3–21](#). These notifications are instances of the Message Queue JMX class `ServiceNotification`, and their names are defined as static constants in that class.

TABLE 3–21 Service Manager Monitor Notifications

Name	Utility Constant	Description
mq.service.pause	<code>ServiceNotification.SERVICE_PAUSE</code>	Service paused
mq.service.resume	<code>ServiceNotification.SERVICE_RESUME</code>	Service resumed

[Table 3–22](#) shows the method defined in class `ServiceNotification` for obtaining details about a service manager monitor notification.

TABLE 3-22 Data Retrieval Method for Service Manager Monitor Notifications

Method	Result Type	Description
getServiceName	String	Service name See Table 3-11 for possible values.

Connections

This section describes the MBeans used for managing connections:

- The connection configuration MBean configures a connection.
- The connection monitor MBean monitors a connection.
- The connection manager configuration MBean manages connection configuration MBeans.
- The connection manager monitor MBean manages connection monitor MBeans.

The following subsections describe each of these MBeans in detail.

Connection Configuration

The *connection configuration MBean* is used for configuring a connection. There is one such MBean for each connection.

Object Name

The connection configuration MBean has an object name of the following form:

```
com.sun.messaging.jms.server:type=Connection,subtype=Config,id=connectionID
```

where *connectionID* is the connection identifier. For example:

```
com.sun.messaging.jms.server:type=Connection,subtype=Config,
id=7853717387765338368
```

The utility class `MQObjectName` provides a static method, `createConnectionConfig`, for constructing object names of this form.

Attribute

The connection configuration MBean has the attribute shown in [Table 3-23](#). The name of this attribute is defined as a static constant in the utility class `ConnectionAttributes`.

TABLE 3-23 Connection Configuration Attribute

Name	Type	Settable?	Description
ConnectionID	String	No	Connection identifier

Connection Monitor

The *connection monitor MBean* is used for monitoring a connection. There is one such MBean for each connection.

Object Name

The connection monitor MBean has an object name of the following form:

```
com.sun.messaging.jms.server:type=Connection,subtype=Monitor,id=connectionID
```

where *connectionID* is the connection identifier. For example:

```
com.sun.messaging.jms.server:type=Connection,subtype=Monitor,
id=7853717387765338368
```

The utility class `MQObjectName` provides a static method, `createConnectionMonitor`, for constructing object names of this form.

Attributes

The connection monitor MBean has the attributes shown in [Table 3-24](#). The names of these attributes are defined as static constants in the utility class `ConnectionAttributes`.

TABLE 3-24 Connection Monitor Attributes

Name	Type	Settable?	Description
ConnectionID	String	No	Connection identifier
Host	String	No	Host from which connection was made
Port	Integer	No	Port number
ServiceName	String	No	Connection service name
User	String	No	User name
ClientID	String	No	Client identifier
ClientPlatform	String	No	String describing client platform

TABLE 3–24 Connection Monitor Attributes (Continued)

Name	Type	Settable?	Description
NumProducers	Integer	No	Current number of associated message producers
NumConsumers	Integer	No	Current number of associated message consumers

Operations

The connection monitor MBean supports the operations shown in Table 3–25. The names of these operations are defined as static constants in the utility class `ConnectionOperations`.

TABLE 3–25 Connection Monitor Operations

Name	Parameters	Result Type	Description
<code>getService</code>	None	<code>ObjectName</code>	Object name of service monitor MBean for associated connection service
<code>getTemporaryDestinations</code>	None	<code>ObjectName[]</code>	Object names of destination monitor MBeans for all associated temporary destinations
<code>getProducerIDs</code>	None	<code>String[]</code>	Producer identifiers of all associated message producers
<code>getConsumerIDs</code>	None	<code>String[]</code>	Consumer identifiers of all associated message consumers

Connection Manager Configuration

Each broker has a single *connection manager configuration MBean*, used for managing all of the broker's connection configuration MBeans.

Object Name

The connection manager configuration MBean has the following object name:

```
com.sun.messaging.jms.server:type=ConnectionManager,subtype=Config
```

A string representing this object name is defined as a static constant `CONNECTION_MANAGER_CONFIG_MBEAN_NAME` in the utility class `MQObjectName`.

Attribute

The connection manager configuration MBean has the attribute shown in Table 3–26. The name of this attribute is defined as a static constant in the utility class `ConnectionAttributes`.

TABLE 3–26 Connection Manager Configuration Attribute

Name	Type	Settable?	Description
NumConnections	Integer	No	Number of current connections

Operations

The connection manager configuration MBean supports the operations shown in [Table 3–27](#). The names of these operations are defined as static constants in the utility class `ConnectionOperations`.

TABLE 3–27 Connection Manager Configuration Operations

Name	Parameters	Result Type	Description
<code>getConnections</code>	None	<code>ObjectName[]</code>	Object names of connection configuration MBeans for all current connections
<code>destroy</code>	<code>connectionID</code> (Long)	None	Destroy connection The desired connection is designated by its connection identifier (<code>connectionID</code>).

Connection Manager Monitor

Each broker has a single *connection manager monitor MBean*, used for managing all of the broker's connection monitor MBeans.

Object Name

The connection manager monitor MBean has the following object name:

```
com.sun.messaging.jms.server:type=ConnectionManager,subtype=Monitor
```

A string representing this object name is defined as a static constant `CONNECTION_MANAGER_MONITOR_MBEAN_NAME` in the utility class `MQObjectName`.

Attributes

The connection manager monitor MBean has the attributes shown in [Table 3–28](#). The names of these attributes are defined as static constants in the utility class `ConnectionAttributes`.

TABLE 3–28 Connection Manager Monitor Attributes

Name	Type	Settable?	Description
NumConnections	Integer	No	Current number of connections
NumConnectionsOpened	Long	No	Cumulative number of connections opened since broker started
NumConnectionsRejected	Long	No	Cumulative number of connections rejected since broker started

Operation

The connection manager monitor MBean supports the operation shown in [Table 3–29](#). The name of this operation is defined as a static constant in the utility class `ConnectionOperations`.

TABLE 3–29 Connection Manager Monitor Operation

Name	Parameters	Result Type	Description
getConnections	None	ObjectName[]	Object names of connection monitor MBeans for all current connections

Notifications

The connection manager monitor MBean supports the notifications shown in [Table 3–30](#). These notifications are instances of the Message Queue JMX class `ConnectionNotification`, and their names are defined as static constants in that class.

TABLE 3–30 Connection Manager Monitor Notifications

Name	Utility Constant	Description
mq.connection.open	<code>ConnectionNotification.CONNECTION_OPEN</code>	Connection opened
mq.connection.reject	<code>ConnectionNotification.CONNECTION_REJECT</code>	Connection rejected
mq.connection.close	<code>ConnectionNotification.CONNECTION_CLOSE</code>	Connection closed

[Table 3–31](#) shows the methods defined in class `ConnectionNotification` for obtaining details about a connection manager monitor notification.

TABLE 3–31 Data Retrieval Methods for Connection Manager Monitor Notifications

Method	Result Type	Description
getConnectionID	String	Connection identifier
getRemoteHost	String	Host from which connection was made
getServiceName	String	Connection service name

TABLE 3–31 Data Retrieval Methods for Connection Manager Monitor Notifications (Continued)

Method	Result Type	Description
getUserName	String	User name

Destinations

This section describes the MBeans used for managing destinations:

- The destination configuration MBean configures a destination.
- The destination monitor MBean monitors a destination.
- The destination manager configuration MBean manages destination configuration MBeans.
- The destination manager monitor MBean manages destination monitor MBeans.

The following subsections describe each of these MBeans in detail.

Destination Configuration

The *destination configuration MBean* is used for configuring a destination. There is one such MBean for each destination.

Object Name

The destination configuration MBean has an object name of the following form:

```
com.sun.messaging.jms.server:type=Destination,subtype=Config,
desttype=destinationType,name=destinationName
```

where *destinationType* is one of the destination types shown in [Table 3–33](#) and *destinationName* is the name of the destination. For example:

```
com.sun.messaging.jms.server:type=Destination,subtype=Config,desttype=t,
name="Dest"
```

The utility class `MQObjectName` provides a static method, `createDestinationConfig`, for constructing object names of this form.

Attributes

The destination configuration MBean has the attributes shown in [Table 3–32](#). The names of these attributes are defined as static constants in the utility class `DestinationAttributes`.

TABLE 3-32 Destination Configuration Attributes

Name	Type	Settable?	Description
Name	String	No	Destination name
Type	String	No	Destination type See Table 3-33 for possible values.
MaxNumMsgs	Long	Yes	Maximum number of unconsumed messages A value of -1 denotes an unlimited number of messages.
MaxBytesPerMsg	Long	Yes	Maximum size, in bytes, of any single message Rejection of a persistent message is reported to the producing client with an exception; no notice is sent for nonpersistent messages. A value of -1 denotes an unlimited message size.
MaxTotalMsgBytes	Long	Yes	Maximum total memory, in bytes, for unconsumed messages
LimitBehavior	String	Yes	Broker behavior when memory-limit threshold reached See Table 3-34 for possible values. If the value is REMOVE_OLDEST or REMOVE_LOW_PRIORITY and the UseDMQ attribute is true, excess messages are moved to the dead message queue.
MaxNumProducers	Integer	Yes	Maximum number of associated message producers When this limit is reached, no new producers can be created. A value of -1 denotes an unlimited number of producers.
MaxNumActiveConsumers ¹	Integer	Yes	Maximum number of associated active message consumers in load-balanced delivery A value of -1 denotes an unlimited number of consumers.
MaxNumBackupConsumers ¹	Integer	Yes	Maximum number of associated backup message consumers in load-balanced delivery A value of -1 denotes an unlimited number of consumers.
ConsumerFlowLimit	Long	Yes	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 consumers.

¹ Queue destinations only

TABLE 3–32 Destination Configuration Attributes (Continued)

Name	Type	Settable?	Description
LocalOnly	Boolean	No	Local delivery only? 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).
LocalDeliveryPreferred ¹	Boolean	Yes	Local delivery preferred? This property applies only to load-balanced delivery in broker clusters. If <code>true</code> , messages will be delivered to remote consumers only if there are no associated consumers on the local broker. The destination must not be restricted to local-only delivery (<code>LocalOnly</code> must be <code>false</code>).
UseDMQ	Boolean	Yes	Send dead messages to dead message queue? If <code>false</code> , dead messages will simply be discarded.

¹ Queue destinations only

Table 3–33 shows the possible values for the `Type` attribute. These values are defined as static constants in the utility class `DestinationType`.

TABLE 3–33 Destination Configuration Type Values

Value	Utility Constant	Meaning
q	<code>DestinationType.QUEUE</code>	Queue (point-to-point) destination
t	<code>DestinationType.TOPIC</code>	Topic (publish/subscribe) destination

Table 3–34 shows the possible values for the `LimitBehavior` attribute. These values are defined as static constants in the utility class `DestinationLimitBehavior`.

TABLE 3–34 Destination Limit Behaviors

Value	Utility Constant	Meaning
FLOW_CONTROL	<code>DestinationLimitBehavior.FLOW_CONTROL</code>	Slow down producers
REMOVE_OLDEST	<code>DestinationLimitBehavior.REMOVE_OLDEST</code>	Throw out oldest messages
REMOVE_LOW_PRIORITY	<code>DestinationLimitBehavior.REMOVE_LOW_PRIORITY</code>	Throw out lowest-priority messages according to age; no notice to producing client

TABLE 3–34 Destination Limit Behaviors (Continued)

Value	Utility Constant	Meaning
REJECT_NEWEST	<code>DestinationLimitBehavior.REJECT_NEWEST</code>	Reject newest messages; notify producing client with an exception only if message is persistent

Operations

The destination configuration MBean supports the operations shown in [Table 3–35](#). The names of these operations are defined as static constants in the utility class `DestinationOperations`.

TABLE 3–35 Destination Configuration Operations

Name	Parameters	Result Type	Description
<code>pause</code>	<code>pauseType</code> (String)	None	Pause message delivery See Table 3–36 for possible values of <code>pauseType</code> .
<code>pause</code>	None	None	Pause all message delivery Equivalent to <code>pause(DestinationPauseType.ALL)</code> .
<code>resume</code>	None	None	Resume message delivery
<code>purge</code>	None	None	Purge all messages
<code>compact</code> ¹	None	None	Compact persistent data store Note – Only a paused destination can be compacted.

¹ File-based persistence only

[Table 3–36](#) shows the possible values for the pause operation's `pauseType` parameter. These values are defined as static constants in the utility class `DestinationPauseType`.

TABLE 3–36 Destination Pause Types

Value	Utility Constant	Meaning
PRODUCERS	<code>DestinationPauseType.PRODUCERS</code>	Pause delivery from associated message producers
CONSUMERS	<code>DestinationPauseType.CONSUMERS</code>	Pause delivery to associated message consumers
ALL	<code>DestinationPauseType.ALL</code>	Pause all message delivery

Notification

The destination configuration MBean supports the notification shown in [Table 3–37](#).

TABLE 3-37 Destination Configuration Notification

Name	Description
<code>jmx.attribute.change</code>	Attribute value changed

Destination Monitor

The *destination monitor MBean* is used for monitoring a destination. There is one such MBean for each destination.

Object Name

The destination monitor MBean has an object name of the following form:

```
com.sun.messaging.jms.server:type=Destination,subtype=Monitor,
desttype=destinationType,name=destinationName
```

where *destinationType* is one of the destination types shown in [Table 3-39](#) and *destinationName* is the name of the destination. For example:

```
com.sun.messaging.jms.server:type=Destination,subtype=Monitor,desttype=t,
name="Dest"
```

The utility class `MQObjectName` provides a static method, `createDestinationMonitor`, for constructing object names of this form.

Attributes

The destination monitor MBean has the attributes shown in [Table 3-38](#). The names of these attributes are defined as static constants in the utility class `DestinationAttributes`.

TABLE 3-38 Destination Monitor Attributes

Name	Type	Settable?	Description
Name	String	No	Destination name
Type	String	No	Destination type See Table 3-39 for possible values.
CreatedByAdmin	Boolean	No	Administrator-created destination?
Temporary	Boolean	No	Temporary destination?

TABLE 3-38 Destination Monitor Attributes (Continued)

Name	Type	Settable?	Description
ConnectionID ¹	String	No	Connection identifier
State	Integer	No	Current state See Table 3-40 for possible values.
StateLabel	String	No	String representation of current state: Useful for displaying the state in human-readable form, such as in the Java Monitoring and Management Console (jconsole). See Table 3-40 for possible values.
NumProducers	Integer	No	Current number of associated message producers
NumConsumers	Integer	No	Current number of associated message consumers 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 NumActiveConsumers.
PeakNumConsumers	Integer	No	Peak number of associated message consumers since broker started 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 PeakNumActiveConsumers.
AvgNumConsumers	Integer	No	Average number of associated message consumers since broker started 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 AvgNumActiveConsumers.
NumActiveConsumers	Integer	No	Current number of associated active message consumers For topic destinations, this attribute includes both nondurable and (active and inactive) durable subscribers and is equivalent to NumConsumers.
PeakNumActiveConsumers	Integer	No	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 PeakNumConsumers.

¹ Temporary destinations only

TABLE 3-38 Destination Monitor Attributes (Continued)

Name	Type	Settable?	Description
AvgNumActiveConsumers	Integer	No	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 AvgNumConsumers.
NumBackupConsumers ²	Integer	No	Current number of associated backup message consumers
PeakNumBackupConsumers ²	Integer	No	Peak number of associated backup message consumers since broker started
AvgNumBackupConsumers ²	Integer	No	Average number of associated backup message consumers since broker started
NumMsgs	Long	No	Current number of messages stored in memory and persistent store Does not include messages held in transactions.
NumMsgsPendingAcks	Long	No	Current number of messages being held in memory and persistent store pending acknowledgment
NumMsgsHeldInTransaction	Long	No	Current number of messages being held in memory and persistent store in uncommitted transactions
PeakNumMsgs	Long	No	Peak number of messages stored in memory and persistent store since broker started
AvgNumMsgs	Long	No	Average number of messages stored in memory and persistent store since broker started
NumMsgsIn	Long	No	Cumulative number of messages received since broker started
NumMsgsOut	Long	No	Cumulative number of messages sent since broker started
MsgBytesIn	Long	No	Cumulative size in bytes of messages received since broker started
MsgBytesOut	Long	No	Cumulative size in bytes of messages sent since broker started
PeakMsgBytes	Long	No	Size in bytes of largest single message received since broker started
TotalMsgBytes	Long	No	Current total size in bytes of messages stored in memory and persistent store Does not include messages held in transactions.
TotalMsgBytesHeldInTransaction	Long	No	Current total size in bytes of messages being held in memory and persistent store in uncommitted transactions

² Queue destinations only

TABLE 3–38 Destination Monitor Attributes (Continued)

Name	Type	Settable?	Description
PeakTotalMsgBytes	Long	No	Peak total size in bytes of messages stored in memory and persistent store since broker started
AvgTotalMsgBytes	Long	No	Average total size in bytes of messages stored in memory and persistent store since broker started
DiskReserved ³	Long	No	Amount of disk space, in bytes, reserved for destination
DiskUsed ³	Long	No	Amount of disk space, in bytes, currently in use by destination
DiskUtilizationRatio ³	Integer	No	Ratio of disk space currently in use to disk space reserved for destination

³ File-based persistence only

Table 3–39 shows the possible values for the Type attribute. These values are defined as static constants in the utility class `DestinationType`.

TABLE 3–39 Destination Monitor Type Values

Value	Utility Constant	Meaning
q	<code>DestinationType.QUEUE</code>	Queue (point-to-point) destination
t	<code>DestinationType.TOPIC</code>	Topic (publish/subscribe) destination

Table 3–40 shows the possible values for the State and StateLabel attributes. These values are defined as static constants in the utility class `DestinationState`.

TABLE 3–40 Destination State Values

Value	Utility Constant	String Representation	Meaning
0	<code>DestinationState.RUNNING</code>	RUNNING	Destination running
1	<code>DestinationState.CONSUMERS_PAUSED</code>	CONSUMERS_PAUSED	Message consumers paused
2	<code>DestinationState.PRODUCERS_PAUSED</code>	PRODUCERS_PAUSED	Message producers paused
3	<code>DestinationState.PAUSED</code>	PAUSED	Destination paused
–1	<code>DestinationState.UNKNOWN</code>	UNKNOWN	Destination state unknown

Operations

The destination monitor MBean supports the operations shown in Table 3–41. The names of these operations are defined as static constants in the utility class `DestinationOperations`.

TABLE 3-41 Destination Monitor Operations

Name	Parameters	Result Type	Description
getConnection ¹	None	ObjectName	Object name of connection monitor MBean for connection
getProducerIDs	None	String[]	Producer identifiers of all current associated message producers
getConsumerIDs	None	String[]	Consumer identifiers of all current associated message consumers For queue destinations, this operation returns both active and backup consumers. For topic destinations, it returns both nondurable and (active and inactive) durable subscribers.
getActiveConsumerIDs	None	String[]	Consumer identifiers of all current associated active message consumers For topic destinations, this operation returns both nondurable and (active and inactive) durable subscribers.
getBackupConsumerIDs ²	None	String[]	Consumer identifiers of all current associated backup message consumers

¹ Temporary destinations only

² Queue destinations only

Notifications

The destination monitor MBean supports the notifications shown in [Table 3-42](#). These notifications are instances of the Message Queue JMX class `DestinationNotification`, and their names are defined as static constants in that class.

TABLE 3-42 Destination Monitor Notifications

Name	Utility Constant	Description
mq.destination.pause	<code>DestinationNotification.DESTINATION_PAUSE</code>	Destination paused
mq.destination.resume	<code>DestinationNotification.DESTINATION_RESUME</code>	Destination resumed
mq.destination.compact	<code>DestinationNotification.DESTINATION_COMPACT</code>	Destination compacted
mq.destination.purge	<code>DestinationNotification.DESTINATION_PURGE</code>	Destination purged

[Table 3-43](#) shows the methods defined in class `DestinationNotification` for obtaining details about a destination monitor notification.

TABLE 3-43 Data Retrieval Methods for Destination Monitor Notifications

Method	Result Type	Description
<code>getDestinationName</code>	String	Destination name
<code>getDestinationType</code>	String	Destination type See Table 3-39 for possible values.
<code>getCreatedByAdmin</code>	Boolean	Administrator-created destination?
<code>getPauseType</code>	String	Pause type See Table 3-36 for possible values.

Destination Manager Configuration

Each broker has a single *destination manager configuration MBean*, used for managing all of the broker's destination configuration MBeans.

Object Name

The destination manager configuration MBean has the following object name:

```
com.sun.messaging.jms.server:type=DestinationManager,subtype=Config
```

A string representing this object name is defined as a static constant `DESTINATION_MANAGER_CONFIG_MBEAN_NAME` in the utility class `MQObjectName`.

Attributes

The destination manager configuration MBean has the attributes shown in [Table 3-44](#). The names of these attributes are defined as static constants in the utility class `DestinationAttributes`.

TABLE 3-44 Destination Manager Configuration Attributes

Name	Type	Settable?	Description
<code>AutoCreateQueues</code>	Boolean	Yes	Allow auto-creation of queue destinations?
<code>AutoCreateTopics</code>	Boolean	Yes	Allow auto-creation of topic destinations?
<code>NumDestinations</code>	Integer	No	Current total number of destinations
<code>MaxNumMsgs</code>	Long	Yes	Maximum total number of unconsumed messages A value of <code>-1</code> denotes an unlimited number of messages.

TABLE 3-44 Destination Manager Configuration Attributes (Continued)

Name	Type	Settable?	Description
MaxBytesPerMsg	Long	Yes	Maximum size, in bytes, of any single message A value of -1 denotes an unlimited message size.
MaxTotalMsgBytes	Long	Yes	Maximum total memory, in bytes, for unconsumed messages A value of -1 denotes an unlimited number of bytes.
AutoCreateQueueMaxNumActiveConsumers ¹	Integer	Yes	Maximum total number of active message consumers in load-balanced delivery A value of -1 denotes an unlimited number of consumers.
AutoCreateQueueMaxNumBackupConsumers ¹	Integer	Yes	Maximum total number of backup message consumers in load-balanced delivery A value of -1 denotes an unlimited number of consumers.
DMQTruncateBody	Boolean	Yes	Remove message body before storing in dead message queue? If <code>true</code> , only the message header and property data will be saved.
LogDeadMsgs	Boolean	Yes	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.

¹ Auto-created queue destinations only

Operations

The destination manager configuration MBean supports the operations shown in [Table 3-45](#). The names of these operations are defined as static constants in the utility class `DestinationOperations`.

TABLE 3-45 Destination Manager Configuration Operations

Name	Parameters	Result Type	Description
getDestinations	None	ObjectName[]	Object names of destination configuration MBeans for all current destinations
create	<i>destinationType</i> (String) <i>destinationName</i> (String) <i>destinationAttributes</i> (AttributeList)	None	Create destination with specified type, name, and attributes The <i>destinationType</i> and <i>destinationName</i> parameters are required, but <i>destinationAttributes</i> may be null. See Table 3-46 for possible values of <i>destinationType</i> . The <i>destinationAttributes</i> list may include any of the attributes listed in Table 3-32 except Name and Type. The names of these attributes are defined as static constants in the utility class <code>DestinationAttributes</code> .
create	<i>destinationType</i> (String) <i>destinationName</i> (String)	None	Create destination with specified type and name Equivalent to <code>create(destinationType, destinationName, null)</code> . See Table 3-46 for possible values of <i>destinationType</i> .
destroy	<i>destinationType</i> (String) <i>destinationName</i> (String)	None	Destroy destination See Table 3-46 for possible values of <i>destinationType</i> .
pause	<i>pauseType</i> (String)	None	Pause message delivery for all destinations See Table 3-47 for possible values of <i>pauseType</i> .
pause	None	None	Pause all message delivery for all destinations Equivalent to <code>pause(DestinationPauseType.ALL)</code> .
resume	None	None	Resume message delivery for all destinations
compact ¹	None	None	Compact all destinations Note – Only paused destinations can be compacted.

¹ File-based persistence only

Table 3-46 shows the possible values for the create and destroy operations' *destinationType* parameters. These values are defined as static constants in the utility class `DestinationType`.

TABLE 3–46 Destination Manager Configuration Type Values

Value	Utility Constant	Meaning
q	<code>DestinationType.QUEUE</code>	Queue (point-to-point) destination
t	<code>DestinationType.TOPIC</code>	Topic (publish/subscribe) destination

Table 3–47 shows the possible values for the pause operation's *pauseType* parameter. These values are defined as static constants in the utility class `DestinationPauseType`.

TABLE 3–47 Destination Manager Pause Types

Value	Utility Constant	Meaning
PRODUCERS	<code>DestinationPauseType.PRODUCERS</code>	Pause delivery from associated message producers
CONSUMERS	<code>DestinationPauseType.CONSUMERS</code>	Pause delivery to associated message consumers
ALL	<code>DestinationPauseType.ALL</code>	Pause all delivery

Notification

The destination manager configuration MBean supports the notification shown in Table 3–48.

TABLE 3–48 Destination Manager Configuration Notification

Name	Description
<code>jmx.attribute.change</code>	Attribute value changed

Destination Manager Monitor

Each broker has a single *destination manager monitor MBean*, used for managing all of the broker's destination monitor MBeans.

Object Name

The destination manager monitor MBean has the following object name:

```
com.sun.messaging.jms.server:type=DestinationManager,subtype=Monitor
```

A string representing this object name is defined as a static constant `DESTINATION_MANAGER_MONITOR_MBEAN_NAME` in the utility class `MQObjectName`.

Attributes

The destination manager monitor MBean has the attributes shown in Table 3–49. The names of these attributes are defined as static constants in the utility class `DestinationAttributes`.

TABLE 3-49 Destination Manager Monitor Attributes

Name	Type	Settable?	Description
NumDestinations	Integer	No	Current total number of destinations
NumMsgs	Long	No	Current total number of messages stored in memory and persistent store for all destinations Does not include messages held in transactions.
TotalMsgBytes	Long	No	Current total size in bytes of messages stored in memory and persistent store for all destinations Does not include messages held in transactions.
NumMsgsInDMQ	Long	No	Current number of messages stored in memory and persistent store for dead message queue
TotalMsgBytesInDMQ	Long	No	Current total size in bytes of messages stored in memory and persistent store for dead message queue

Operation

The destination manager monitor MBean supports the operation shown in [Table 3-50](#). The name of this operation is defined as a static constant in the utility class `DestinationOperations`.

TABLE 3-50 Destination Manager Monitor Operation

Name	Parameters	Result Type	Description
getDestinations	None	ObjectName[]	Object names of destination monitor MBeans for all current destinations

Notifications

The destination manager monitor MBean supports the notifications shown in [Table 3-51](#). These notifications are instances of the Message Queue JMX class `DestinationNotification`, and their names are defined as static constants in that class.

TABLE 3-51 Destination Manager Monitor Notifications

Name	Utility Constant	Description
mq.destination.create	<code>DestinationNotification.DESTINATION_CREATE</code>	Destination created
mq.destination.destroy	<code>DestinationNotification.DESTINATION_DESTROY</code>	Destination destroyed
mq.destination.pause	<code>DestinationNotification.DESTINATION_PAUSE</code>	Destination paused

TABLE 3-51 Destination Manager Monitor Notifications (Continued)

Name	Utility Constant	Description
mq.destination.resume	DestinationNotification.DESTINATION_RESUME	Destination resumed
mq.destination.compact	DestinationNotification.DESTINATION_COMPACT	Destination compacted
mq.destination.purge	DestinationNotification.DESTINATION_PURGE	Destination purged

Table 3-52 shows the methods defined in class `DestinationNotification` for obtaining details about a destination manager monitor notification.

TABLE 3-52 Data Retrieval Methods for Destination Manager Monitor Notifications

Method	Result Type	Description
<code>getDestinationName</code>	String	Destination name
<code>getDestinationType</code>	String	Destination type See Table 3-46 for possible values.
<code>getCreatedByAdmin</code>	Boolean	Administrator-created destination?
<code>getPauseType</code>	String	Pause type See Table 3-47 for possible values.

Message Producers

This section describes the MBeans used for managing message producers:

- The producer manager configuration MBean configures message producers.
- The producer manager monitor MBean monitors message producers.

The following subsections describe each of these MBeans in detail.

Note – Notice that there are no resource MBeans associated with individual message producers; rather, all producers are managed through the broker's global producer manager configuration and producer manager monitor MBeans.

Producer Manager Configuration

Each broker has a single *producer manager configuration MBean*, used for configuring all of the broker's message producers.

Object Name

The producer manager configuration MBean has the following object name:

```
com.sun.messaging.jms.server:type=ProducerManager,subtype=Config
```

A string representing this object name is defined as a static constant `PRODUCER_MANAGER_CONFIG_MBEAN_NAME` in the utility class `MQObjectName`.

Attribute

The producer manager configuration MBean has the attribute shown in [Table 3–53](#). The name of this attribute is defined as a static constant in the utility class `ProducerAttributes`.

TABLE 3–53 Producer Manager Configuration Attribute

Name	Type	Settable?	Description
NumProducers	Integer	No	Current total number of message producers

Operation

The producer manager configuration MBean supports the operation shown in [Table 3–54](#). The name of this operation is defined as a static constant in the utility class `ProducerOperations`.

TABLE 3–54 Producer Manager Configuration Operation

Name	Parameters	Result Type	Description
getProducerIDs	None	String[]	Producer identifiers of all current message producers

Producer Manager Monitor

Each broker has a single *producer manager monitor MBean*, used for monitoring all of the broker's message producers.

Object Name

The producer manager monitor MBean has the following object name:

```
com.sun.messaging.jms.server:type=ProducerManager,subtype=Monitor
```

A string representing this object name is defined as a static constant `PRODUCER_MANAGER_MONITOR_MBEAN_NAME` in the utility class `MQObjectName`.

Attribute

The producer manager monitor MBean has the attribute shown in [Table 3–55](#). The name of this attribute is defined as a static constant in the utility class `ProducerAttributes`.

TABLE 3–55 Producer Manager Monitor Attribute

Name	Type	Settable?	Description
<code>NumProducers</code>	Integer	No	Current total number of message producers

Operations

The producer manager monitor MBean supports the operations shown in [Table 3–56](#). The names of these operations are defined as static constants in the utility class `ProducerOperations`.

TABLE 3–56 Producer Manager Monitor Operations

Name	Parameters	Result Type	Description
<code>getProducerIDs</code>	None	<code>String[]</code>	Producer identifiers of all current message producers
<code>getProducerInfoByID</code>	<i>producerID</i> (<code>String</code>)	<code>CompositeData</code>	Descriptive information about message producer The desired producer is designated by its producer identifier (<i>producerID</i>). The value returned is a <code>JMX CompositeData</code> object describing the producer; see Table 3–57 for lookup keys used with this object.
<code>getProducerInfo</code>	None	<code>CompositeData[]</code>	Descriptive information about all current message producers The value returned is an array of <code>JMX CompositeData</code> objects describing the producers; see Table 3–57 for lookup keys used with these objects.

The `getProducerInfoByID` and `getProducerInfo` operations return objects implementing the `JMX CompositeData` interface, which maps lookup keys to associated data values. The keys shown in [Table 3–57](#) are defined as static constants in the utility class `ProducerInfo` for use with these objects.

TABLE 3-57 Lookup Keys for Message Producer Information

Name	ValueType	Description
ProducerID	String	Producer identifier
ServiceName	String	Name of associated connection service
ConnectionID	String	Connection identifier of associated connection
Host	String	Connection's host name
User	String	Connection's user name
DestinationName	String	Name of associated destination
DestinationType	String	Type of associated destination See Table 3-58 for possible values.
FlowPaused	Boolean	Message delivery paused?
NumMsgs	Long	Number of messages sent

Table 3-58 shows the possible values returned for the lookup key `DestinationType`. These values are defined as static constants in the utility class `DestinationType`.

TABLE 3-58 Message Producer Destination Types

Value	Utility Constant	Meaning
q	<code>DestinationType.QUEUE</code>	Queue (point-to-point) destination
t	<code>DestinationType.TOPIC</code>	Topic (publish/subscribe) destination

Message Consumers

This section describes the MBeans used for managing message consumers:

- The consumer manager configuration MBean configures message consumers.
- The consumer manager monitor MBean monitors message consumers.

The following subsections describe each of these MBeans in detail.

Note – Notice that there are no resource MBeans associated with individual message consumers; rather, all consumers are managed through the broker's global consumer manager configuration and consumer manager monitor MBeans.

Consumer Manager Configuration

Each broker has a single *consumer manager configuration MBean*, used for configuring all of the broker's message consumers.

Object Name

The consumer manager configuration MBean has the following object name:

```
com.sun.messaging.jms.server:type=ConsumerManager,subtype=Config
```

A string representing this object name is defined as a static constant `CONSUMER_MANAGER_CONFIG_MBEAN_NAME` in the utility class `MQObjectName`.

Attribute

The consumer manager configuration MBean has the attribute shown in [Table 3–59](#). The name of this attribute is defined as a static constant in the utility class `ConsumerAttributes`.

TABLE 3–59 Consumer Manager Configuration Attribute

Name	Type	Settable?	Description
NumConsumers	Integer	No	Current total number of message consumers

Operations

The consumer manager configuration MBean supports the operations shown in [Table 3–60](#). The names of these operations are defined as static constants in the utility class `ConsumerOperations`.

TABLE 3–60 Consumer Manager Configuration Operations

Name	Parameters	Result Type	Description
getConsumerIDs	None	String[]	Consumer identifiers of all current message consumers
purge ¹	<i>consumerID</i> (String)	None	Purge all messages The desired subscriber is designated by its consumer identifier (<i>consumerID</i>). The subscriber itself is not destroyed.

¹ Durable topic subscribers only

Consumer Manager Monitor

Each broker has a single *consumer manager monitor MBean*, used for monitoring all of the broker's message consumers.

Object Name

The consumer manager monitor MBean has the following object name:

```
com.sun.messaging.jms.server:type=ConsumerManager,subtype=Monitor
```

A string representing this object name is defined as a static constant `CONSUMER_MANAGER_MONITOR_MBEAN_NAME` in the utility class `MQObjectName`.

Attribute

The consumer manager monitor MBean has the attribute shown in [Table 3–61](#). The name of this attribute is defined as a static constant in the utility class `ConsumerAttributes`.

TABLE 3–61 Consumer Manager Monitor Attribute

Name	Type	Settable?	Description
NumConsumers	Integer	No	Current total number of message consumers

Operations

The consumer manager monitor MBean supports the operations shown in [Table 3–62](#). The names of these operations are defined as static constants in the utility class `ConsumerOperations`.

TABLE 3–62 Consumer Manager Monitor Operations

Name	Parameters	Result Type	Description
getConsumerIDs	None	String[]	Consumer identifiers of all current message consumers
getConsumerInfoByID	<i>consumerID</i> (String)	CompositeData	Descriptive information about message consumer The desired consumer is designated by its consumer identifier (<i>consumerID</i>). The value returned is a JMX <code>CompositeData</code> object describing the consumer; see Table 3–63 for lookup keys used with this object.

TABLE 3-62 Consumer Manager Monitor Operations (Continued)

Name	Parameters	Result Type	Description
getConsumerInfo	None	CompositeData[]	<p>Descriptive information about all current message consumers</p> <p>The value returned is an array of JMX CompositeData objects describing the consumers; see Table 3-63 for lookup keys used with these objects.</p>

The getConsumerInfoByID and getConsumerInfo operations return objects implementing the JMX interface CompositeData, which maps lookup keys to associated data values. The keys shown in Table 3-63 are defined as static constants in the utility class ConsumerInfo for use with these objects.

TABLE 3-63 Lookup Keys for Message Consumer Information

Name	Value Type	Description
ConsumerID	String	Consumer identifier
Selector	String	Message selector
ServiceName	String	Name of associated connection service
ConnectionID	String	Connection identifier of associated connection
Host	String	Connection's host name
User	String	Connection's user name
DestinationName	String	Name of associated destination
DestinationType	String	Type of associated destination See Table 3-64 for possible values.
AcknowledgeMode	Integer	Acknowledgment mode of associated session See Table 3-65 for possible values.
AcknowledgeModeLabel	String	String representation of acknowledgment mode Useful for displaying the acknowledgment mode in human-readable form, such as in the Java Monitoring and Management Console (jconsole). See Table 3-65 for possible values.
Durable	Boolean	Durable topic subscriber?
DurableName ¹	String	Subscription name

¹ Durable topic subscribers only

TABLE 3-63 Lookup Keys for Message Consumer Information (Continued)

Name	Value Type	Description
ClientID ¹	String	Client identifier
DurableActive ¹	Boolean	Subscriber active?
FlowPaused	Boolean	Message delivery paused?
NumMsgs	Long	Cumulative number of messages received
NumMsgsPendingAcks	Long	Current number of messages being held in memory and persistent store pending acknowledgment
LastAckTime	Long	Time of last acknowledgment, in standard Java format (milliseconds since January 1, 1970, 00:00:00 UTC)

¹ Durable topic subscribers only

Table 3-64 shows the possible values returned for the lookup key `DestinationType`. These values are defined as static constants in the utility class `DestinationType`.

TABLE 3-64 Message Consumer Destination Types

Value	Utility Constant	Meaning
q	<code>DestinationType.QUEUE</code>	Queue (point-to-point) destination
t	<code>DestinationType.TOPIC</code>	Topic (publish/subscribe) destination

Table 3-65 shows the possible values returned for the lookup keys `AcknowledgeMode` and `AcknowledgeModeLabel`. Four of these values are defined as static constants in the standard JMS interface `javax.jms.Session`; the fifth (`NO_ACKNOWLEDGE`) is defined in the extended Message Queue version of the interface, `com.sun.messaging.jms.Session`.

TABLE 3-65 Acknowledgment Modes

Value	Utility Constant	String Representation	Meaning
1	<code>javax.jms.Session.AUTO_ACKNOWLEDGE</code>	<code>AUTO_ACKNOWLEDGE</code>	Auto-acknowledge mode
2	<code>javax.jms.Session.CLIENT_ACKNOWLEDGE</code>	<code>CLIENT_ACKNOWLEDGE</code>	Client-acknowledge mode
3	<code>javax.jms.Session.DUPS_OK_ACKNOWLEDGE</code>	<code>DUPS_OK_ACKNOWLEDGE</code>	Dups-OK-acknowledge mode
32768	<code>com.sun.messaging.jms.Session.NO_ACKNOWLEDGE</code>	<code>NO_ACKNOWLEDGE</code>	No-acknowledge mode
0	<code>javax.jms.Session.SESSION_TRANSACTED</code>	<code>SESSION_TRANSACTED</code>	Session is transacted (acknowledgment mode ignored)

Transactions

This section describes the MBeans used for managing transactions:

- The transaction manager configuration MBean configures transactions.
- The transaction manager monitor MBean monitors transactions.

The following subsections describe each of these MBeans in detail.

Note – Notice that there are no resource MBeans associated with individual transactions; rather, all transactions are managed through the broker's global transaction manager configuration and transaction manager monitor MBeans.

Transaction Manager Configuration

Each broker has a single *transaction manager configuration MBean*, used for configuring all of the broker's transactions.

Object Name

The transaction manager configuration MBean has the following object name:

```
com.sun.messaging.jms.server:type=TransactionManager,subtype=Config
```

A string representing this object name is defined as a static constant `TRANSACTION_MANAGER_CONFIG_MBEAN_NAME` in the utility class `MQObjectName`.

Attribute

The transaction manager configuration MBean has the attribute shown in [Table 3–66](#). The name of this attribute is defined as a static constant in the utility class `TransactionAttributes`.

TABLE 3–66 Transaction Manager Configuration Attribute

Name	Type	Settable?	Description
NumTransactions	Integer	No	Current number of open transactions

Operations

The transaction manager configuration MBean supports the operations shown in [Table 3–67](#). The names of these operations are defined as static constants in the utility class `TransactionOperations`.

TABLE 3-67 Transaction Manager Configuration Operations

Name	Parameters	Result Type	Description
getTransactionIDs	None	String[]	Transaction identifiers of all current open transactions
commit	<i>transactionID</i> (String)	None	Commit transaction The desired transaction is designated by its transaction identifier (<i>transactionID</i>).
rollback	<i>transactionID</i> (String)	None	Roll back transaction The desired transaction is designated by its transaction identifier (<i>transactionID</i>).

Transaction Manager Monitor

Each broker has a single *transaction manager monitor MBean*, used for monitoring all of the broker's transactions.

Object Name

The transaction manager monitor MBean has the following object name:

```
com.sun.messaging.jms.server:type=TransactionManager,subtype=Monitor
```

A string representing this object name is defined as a static constant `TRANSACTION_MANAGER_MONITOR_MBEAN_NAME` in the utility class `MQObjectName`.

Attributes

The transaction manager monitor MBean has the attributes shown in [Table 3-68](#). The names of these attributes are defined as static constants in the utility class `TransactionAttributes`.

TABLE 3-68 Transaction Manager Monitor Attributes

Name	Type	Settable?	Description
<code>NumTransactions</code>	Integer	No	Current number of open transactions
<code>NumTransactionsCommitted</code>	Long	No	Cumulative number of transactions committed since broker started
<code>NumTransactionsRollback</code>	Long	No	Cumulative number of transactions rolled back since broker started

Operations

The transaction manager monitor MBean supports the operations shown in [Table 3–69](#). The names of these operations are defined as static constants in the utility class `TransactionOperations`.

TABLE 3–69 Transaction Manager Monitor Operations

Name	Parameters	Result Type	Description
<code>getTransactionIDs</code>	None	<code>String[]</code>	Transaction identifiers of all current open transactions
<code>getTransactionInfoByID</code>	<i>transactionID</i> (String)	<code>CompositeData</code>	Descriptive information about transaction The desired transaction is designated by its transaction identifier (<i>transactionID</i>). The value returned is a JMX <code>CompositeData</code> object describing the transaction; see Table 3–70 for lookup keys used with this object.
<code>getTransactionInfo</code>	None	<code>CompositeData[]</code>	Descriptive information about all current open transactions The value returned is an array of JMX <code>CompositeData</code> objects describing the transactions; see Table 3–70 for lookup keys used with these objects.

The `getTransactionInfoByID` and `getTransactionInfo` operations return objects implementing the JMX interface `CompositeData`, which maps lookup keys to associated data values. The keys shown in [Table 3–70](#) are defined as static constants in the utility class `TransactionInfo` for use with these objects.

TABLE 3–70 Lookup Keys for Transaction Information

Name	Value Type	Description
<code>TransactionID</code>	String	Transaction identifier
<code>XID</code> ¹	String	Distributed transaction identifier (XID)
<code>User</code>	String	User name
<code>ClientID</code>	String	Client identifier
<code>ConnectionString</code>	String	Connection string
<code>CreationTime</code>	Long	Time created, in standard Java format (milliseconds since January 1, 1970, 00:00:00 UTC)

¹ Distributed transactions only

TABLE 3-70 Lookup Keys for Transaction Information (Continued)

Name	Value Type	Description
State	Integer	Current state See Table 3-71 for possible values.
StateLabel	String	String representation of current state Useful for displaying the state in human-readable form, such as in the Java Monitoring and Management Console (<code>jconsole</code>). See Table 3-71 for possible values.
NumMsgs	Long	Number of messages
NumAcks	Long	Number of acknowledgments

Table 3-71 shows the possible values returned for the lookup keys `State` and `StateLabel`. These values are defined as static constants in the utility class `TransactionState`.

TABLE 3-71 Transaction State Values

Value	Utility Constant	String Representation	Meaning
0	<code>TransactionState.CREATED</code>	CREATED	Transaction created
1	<code>TransactionState.STARTED</code>	STARTED	Transaction started
2	<code>TransactionState.FAILED</code>	FAILED	Transaction has failed
3	<code>TransactionState.INCOMPLETE</code>	INCOMPLETE	Transaction incomplete
4	<code>TransactionState.COMPLETE</code>	COMPLETE	Transaction complete
5	<code>TransactionState.PREPARED</code>	PREPARED	Transaction in prepared state ¹
6	<code>TransactionState.COMMITTED</code>	COMMITTED	Transaction committed
7	<code>TransactionState.ROLLEDBACK</code>	ROLLEDBACK	Transaction rolled back
8	<code>TransactionState.TIMED_OUT</code>	TIMED_OUT	Transaction has timed out
-1	<code>TransactionState.UNKNOWN</code>	UNKNOWN	Transaction state unknown

¹ Distributed transactions only

Notifications

The transaction manager monitor MBean supports the notifications shown in Table 3-72. These notifications are instances of the Message Queue JMX class `TransactionNotification`, and their names are defined as static constants in that class.

TABLE 3-72 Transaction Manager Monitor Notifications

Name	Utility Constant	Description
<code>mq.transaction.prepare¹</code>	<code>TransactionNotification.TRANSACTION_PREPARE</code>	Transaction has entered prepared state
<code>mq.transaction.commit</code>	<code>TransactionNotification.TRANSACTION_COMMIT</code>	Transaction committed
<code>mq.transaction.rollback</code>	<code>TransactionNotification.TRANSACTION_ROLLBACK</code>	Transaction rolled back

¹ Distributed transactions only

Table 3-73 shows the method defined in class `TransactionNotification` for obtaining details about a transaction manager monitor notification.

TABLE 3-73 Data Retrieval Method for Transaction Manager Monitor Notifications

Method	Result Type	Description
<code>getTransactionID</code>	String	Transaction identifier

Broker Clusters

This section describes the MBeans used for managing broker clusters:

- The cluster configuration MBean configures a broker's cluster-related properties.
- The cluster monitor MBean monitors the brokers in a cluster.

The following subsections describe each of these MBeans in detail.

Cluster Configuration

The *cluster configuration MBean* is used for configuring a broker's cluster-related properties. There is one such MBean for each broker.

Object Name

The cluster configuration MBean has the following object name:

```
com.sun.messaging.jms.server:type=Cluster,subtype=Config
```

A string representing this object name is defined as a static constant `CLUSTER_CONFIG_MBEAN_NAME` in the utility class `MQObjectName`.

Attributes

The cluster configuration MBean has the attributes shown in Table 3-74. The names of these attributes are defined as static constants in the utility class `ClusterAttributes`.

TABLE 3-74 Cluster Configuration Attributes

Name	Type	Settable?	Description
HighlyAvailable	Boolean	No	High-availability (HA) cluster?
ClusterID ¹	String	No	<p>Cluster identifier</p> <p>Must be unique; no two running clusters may have the same cluster identifier.</p> <p>This identifier is appended to the names of all database tables in the cluster's shared persistent store. Must be an alphanumeric string of no more than $n - 13$ characters, where n is the maximum table name length allowed by the database.</p> <p>Note – For brokers belonging to an HA cluster, this attribute is used in database table names in place of the <code>BrokerID</code> (see Table 3-1).</p>
ConfigFileURL ²	String	Yes	URL of cluster configuration file
LocalBrokerInfo	CompositeData	No	<p>Descriptive information about local broker</p> <p>The value returned is a <code>JMX CompositeData</code> object describing the broker; see Table 3-76 for lookup keys used with this object.</p>
MasterBrokerInfo ²	CompositeData	No	<p>Descriptive information about master broker</p> <p>The value returned is a <code>JMX CompositeData</code> object describing the master broker; see Table 3-76 for lookup keys used with this object.</p>

¹ HA clusters only

² Conventional clusters only

Operations

The cluster configuration MBean supports the operations shown in [Table 3-75](#). The names of these operations are defined as static constants in the utility class `ClusterOperations`.

TABLE 3-75 Cluster Configuration Operations

Name	Parameters	Result Type	Description
getBrokerAddresses	None	String[]	<p>Addresses of brokers in cluster</p> <p>Each address specifies the host name and Port Mapper port number of a broker in the cluster, in the form <i>hostName:portNumber</i>.</p> <p>Example: host1:3000</p> <p>For conventional clusters, the list includes all brokers specified by the broker property <code>imq.cluster.brokerList</code>. For HA clusters, it includes all active and inactive brokers in the cluster table stored in the HA database.</p>
getBrokerIDs ¹	None	String[]	<p>Broker identifiers of brokers in cluster</p> <p>The list includes all active and inactive brokers in the cluster table stored in the HA database.</p>
getBrokerInfoByAddress	<i>brokerAddress</i> (String)	CompositeData	<p>Descriptive information about broker</p> <p>The desired broker is designated by its host name and Port Mapper port number (<i>brokerAddress</i>), in the form <i>hostName:portNumber</i>. The value returned is a JMX <code>CompositeData</code> object describing the broker; see Table 3-76 for lookup keys used with this object.</p>
getBrokerInfoByID ¹	<i>brokerID</i> (String)	CompositeData	<p>Descriptive information about broker</p> <p>The desired broker is designated by its broker identifier (<i>brokerID</i>). The value returned is a JMX <code>CompositeData</code> object describing the broker; see Table 3-76 for lookup keys used with this object. For conventional clusters, the operation returns <code>null</code>.</p>
getBrokerInfo	None	CompositeData[]	<p>Descriptive information about all brokers in cluster</p> <p>The value returned is an array of JMX <code>CompositeData</code> objects describing the brokers; see Table 3-76 for lookup keys used with these objects.</p> <p>For conventional clusters, the array includes all brokers specified by the broker property <code>imq.cluster.brokerList</code>. For HA clusters, it includes all active and inactive brokers in the cluster table stored in the HA database.</p>

¹ HA clusters only

TABLE 3-75 Cluster Configuration Operations (Continued)

Name	Parameters	Result Type	Description
reload ²	None	None	Reload cluster configuration file

² Conventional clusters only

The `LocalBrokerInfo` and `MasterBrokerInfo` attributes and the `getBrokerInfoByAddress`, `getBrokerInfoByID`, and `getBrokerInfo` operations return objects implementing the JMX interface `CompositeData`, which maps lookup keys to associated data values. The keys shown in [Table 3-76](#) are defined as static constants in the utility class `BrokerClusterInfo` for use with these objects.

TABLE 3-76 Lookup Keys for Cluster Configuration Information

Key	Value Type	Description
Address	String	Broker address, in the form <i>hostName:portNumber</i> Example: host1:3000
ID ¹	String	Broker identifier

¹ HA clusters only

Notification

The cluster configuration MBean supports the notification shown in [Table 3-77](#).

TABLE 3-77 Cluster Configuration Notification

Name	Description
<code>jmx.attribute.change</code>	Attribute value changed

Cluster Monitor

The *cluster monitor MBean* is used for monitoring the brokers in a cluster. There is one such MBean for each broker.

Object Name

The cluster monitor MBean has the following object name:

```
com.sun.messaging.jms.server:type=Cluster,subtype=Monitor
```

A string representing this object name is defined as a static constant `CLUSTER_MONITOR_MBEAN_NAME` in the utility class `MQObjectName`.

Attributes

The cluster monitor MBean has the attributes shown in [Table 3–78](#). The names of these attributes are defined as static constants in the utility class `ClusterAttributes`.

TABLE 3–78 Cluster Monitor Attributes

Name	Type	Settable?	Description
<code>HighlyAvailable</code>	Boolean	No	High-availability (HA) cluster?
<code>ClusterID</code> ¹	String	No	Cluster identifier Must be unique; no two running clusters may have the same cluster identifier. This identifier is appended to the names of all database tables in the cluster's shared persistent store. Must be an alphanumeric string of no more than $n - 13$ characters, where n is the maximum table name length allowed by the database. Note – For brokers belonging to an HA cluster, this attribute is used in database table names in place of the <code>BrokerID</code> (see Table 3–4).
<code>ConfigFileURL</code> ²	String	Yes	URL of cluster configuration file
<code>LocalBrokerInfo</code>	CompositeData	No	Descriptive information about local broker The value returned is a <code>JMX CompositeData</code> object describing the broker; see Table 3–80 for lookup keys used with this object.
<code>MasterBrokerInfo</code> ²	CompositeData	No	Descriptive information about master broker The value returned is a <code>JMX CompositeData</code> object describing the master broker; see Table 3–80 for lookup keys used with this object.

¹ HA clusters only

² Conventional clusters only

Operations

The cluster monitor MBean supports the operations shown in [Table 3–79](#). The names of these operations are defined as static constants in the utility class `ClusterOperations`.

TABLE 3-79 Cluster Monitor Operations

Name	Parameters	Result Type	Description
getBrokerAddresses	None	String[]	<p>Addresses of brokers in cluster</p> <p>Each address specifies the host name and Port Mapper port number of a broker in the cluster, in the form <i>hostName:portNumber</i>.</p> <p>Example: host1:3000</p> <p>For conventional clusters, the list includes all brokers specified by the broker property <code>imq.cluster.brokerList</code>. For HA clusters, it includes all active and inactive brokers in the cluster table stored in the HA database.</p>
getBrokerIDs ¹	None	String[]	<p>Broker identifiers of brokers in cluster</p> <p>The list includes all active and inactive brokers in the cluster table stored in the HA database.</p>
getBrokerInfoByAddress	<i>brokerAddress</i> (String)	CompositeData	<p>Descriptive information about broker</p> <p>The desired broker is designated by its host name and Port Mapper port number (<i>brokerAddress</i>), in the form <i>hostName:portNumber</i>. The value returned is a JMX <code>CompositeData</code> object describing the broker; see Table 3-80 for lookup keys used with this object.</p>
getBrokerInfoByID ¹	<i>brokerID</i> (String)	CompositeData	<p>Descriptive information about broker</p> <p>The desired broker is designated by its broker identifier (<i>brokerID</i>). The value returned is a JMX <code>CompositeData</code> object describing the broker; see Table 3-80 for lookup keys used with this object. For conventional clusters, the operation returns <code>null</code>.</p>
getBrokerInfo	None	CompositeData[]	<p>Descriptive information about all brokers in cluster</p> <p>The value returned is an array of JMX <code>CompositeData</code> objects describing the brokers; see Table 3-80 for lookup keys used with these objects.</p> <p>For conventional clusters, the array includes all brokers specified by the broker property <code>imq.cluster.brokerList</code>. For HA clusters, it includes all active and inactive brokers in the cluster table stored in the HA database.</p>

¹ HA clusters only

The `LocalBrokerInfo` and `MasterBrokerInfo` attributes and the `getBrokerInfoByAddress`, `getBrokerInfoByID`, and `getBrokerInfo` operations return objects implementing the JMX interface `CompositeData`, which maps lookup keys to associated data values. The keys shown in [Table 3–80](#) are defined as static constants in the utility class `BrokerClusterInfo` for use with these objects.

TABLE 3–80 Lookup Keys for Cluster Monitor Information

Key	Value Type	Description
Address	String	Broker address, in the form <i>hostName:portNumber</i> Example: host1:3000
ID ¹	String	Broker identifier
State	Integer	Current state of broker See Table 3–81 for possible values.
StateLabel	String	String representation of current broker state Useful for displaying the state in human-readable form, such as in the Java Monitoring and Management Console (<code>jconsole</code>). See Table 3–81 for possible values.
TakeoverBrokerID ¹	String	Broker identifier of broker that has taken over this broker's persistent data store
NumMsgs ¹	Long	Current number of messages stored in memory and persistent store
StatusTimestamp ¹	Long	Time of last status update, in standard Java format (milliseconds since January 1, 1970, 00:00:00 UTC) Used to determine whether a broker is running. The interval at which a broker updates its status can be configured with the broker property <code>imq.cluster.monitor.interval</code> .

¹ HA clusters only

[Table 3–81](#) shows the possible values returned for the lookup keys `State` and `StateLabel`. These values are defined as static constants in the utility class `BrokerState`.

TABLE 3–81 Broker State Values

Value	Utility Constant	String Representation	Meaning
0	<code>BrokerState.OPERATING</code>	OPERATING	Broker is operating

TABLE 3–81 Broker State Values (Continued)

Value	Utility Constant	String Representation	Meaning
1	BrokerState.TAKEOVER_STARTED	TAKEOVER_STARTED	Broker has begun taking over persistent data store from another broker
2	BrokerState.TAKEOVER_COMPLETE	TAKEOVER_COMPLETE	Broker has finished taking over persistent data store from another broker
3	BrokerState.TAKEOVER_FAILED	TAKEOVER_FAILED	Attempted takeover has failed
4	BrokerState.QUIESCE_STARTED	QUIESCE_STARTED	Broker has begun quiescing
5	BrokerState.QUIESCE_COMPLETE	QUIESCE_COMPLETE	Broker has finished quiescing
6	BrokerState.SHUTDOWN_STARTED	SHUTDOWN_STARTED	Broker has begun shutting down
7	BrokerState.BROKER_DOWN	BROKER_DOWN	Broker is down
-1	BrokerState.UNKNOWN	UNKNOWN	Broker state unknown

Notifications

The cluster monitor MBean supports the notifications shown in [Table 3–82](#). These notifications are instances of the Message Queue JMX classes `ClusterNotification` and `BrokerNotification`, and their names are defined as static constants in those classes.

TABLE 3–82 Cluster Monitor Notifications

Name	Utility Constant	Description
<code>mq.cluster.broker.join</code>	<code>ClusterNotification.CLUSTER_BROKER_JOIN</code>	A broker has joined the cluster
<code>mq.cluster.broker.down</code>	<code>ClusterNotification.CLUSTER_BROKER_DOWN</code>	A broker in the cluster has shut down or crashed
<code>mq.broker.takeover.start</code> ¹	<code>BrokerNotification.BROKER_TAKEOVER_START</code>	A broker has begun taking over persistent data store from another broker
<code>mq.broker.takeover.complete</code>	<code>BrokerNotification.BROKER_TAKEOVER_COMPLETE</code>	A broker has finished taking over persistent data store from another broker
<code>mq.broker.takeover.fail</code> ¹	<code>BrokerNotification.BROKER_TAKEOVER_FAIL</code>	An attempted takeover has failed

¹ HA clusters only

[Table 3–83](#) shows the methods defined in class `ClusterNotification` for obtaining details about a cluster monitor notification. See [Table 3–6](#) for the corresponding methods of class `BrokerNotification`.

TABLE 3-83 Data Retrieval Methods for Cluster Monitor Notifications

Method	Result Type	Description
<code>isHighlyAvailable</code>	Boolean	High-availability (HA) cluster?
<code>getClusterID</code>	String	Cluster identifier
<code>getBrokerID</code>	String	Broker identifier of affected broker
<code>getBrokerAddress</code>	String	Address of affected broker, in the form <i>hostName:portNumber</i> Example: <code>host1:3000</code>
<code>isMasterBroker</code> ¹	Boolean	Master broker affected?

¹ Conventional clusters only

Logging

This section describes the MBeans used for logging Message Queue operations:

- The log configuration MBean configures Message Queue logging.
- The log monitor MBean monitors Message Queue logging.

The following subsections describe each of these MBeans in detail.

Log Configuration

Each broker has a single *log configuration MBean*, used for configuring Message Queue logging.

Object Name

The log configuration MBean has the following object name:

```
com.sun.messaging.jms.server:type=Log,subtype=Config
```

A string representing this object name is defined as a static constant `LOG_CONFIG_MBEAN_NAME` in the utility class `MQObjectName`.

Attributes

The log configuration MBean has the attributes shown in [Table 3-84](#). The names of these attributes are defined as static constants in the utility class `LogAttributes`.

TABLE 3–84 Log Configuration Attributes

Name	Type	Settable?	Description
Level	String	Yes	Logging level Specifies the categories of logging information that can be written to an output channel. See Table 3–85 for possible values.
RolloverBytes	Long	Yes	File length, in bytes, at which output rolls over to a new log file A value of –1 denotes an unlimited number of bytes (no rollover based on file length).
RolloverSecs	Long	Yes	Age of file, in seconds, at which output rolls over to a new log file A value of –1 denotes an unlimited number of seconds (no rollover based on file age).

[Table 3–85](#) shows the possible values for the `Level` attribute. Each level includes those above it (for example, `WARNING` includes `ERROR`). These values are defined as static constants in the utility class `LogLevel`.

TABLE 3–85 Log Configuration Logging Levels

Name	Utility Constant	Meaning
NONE	<code>LogLevel.NONE</code>	No logging
ERROR	<code>LogLevel.ERROR</code>	Log error messages
WARNING	<code>LogLevel.WARNING</code>	Log warning messages
INFO	<code>LogLevel.INFO</code>	Log informational messages
UNKNOWN	<code>LogLevel.UNKNOWN</code>	Logging level unknown

Notification

The log configuration MBean supports the notification shown in [Table 3–86](#).

TABLE 3–86 Log Configuration Notification

Name	Description
<code>jmx.attribute.change</code>	Attribute value changed

Log Monitor

Each broker has a single *log monitor MBean*, used for monitoring Message Queue logging.

Object Name

The log monitor MBean has the following object name:

```
com.sun.messaging.jms.server:type=Log,subtype=Monitor
```

A string representing this object name is defined as a static constant `LOG_MONITOR_MBEAN_NAME` in the utility class `MQObjectName`.

Notifications

The log monitor MBean supports the notifications shown in [Table 3–87](#). These notifications are instances of the Message Queue JMX class `LogNotification`, and their names are defined as static utility constants in that class.

Note – A notification listener registered for a particular logging level will receive notifications only for that level and not for those above or below it: for example, a listener registered for the notification `mq.log.level.WARNING` will be notified only of `WARNING` messages and not `ERROR` or `INFO`. To receive notifications for more than one logging level, the listener must be explicitly registered for each level separately.

TABLE 3–87 Log Monitor Notifications

Name	Utility Constant	Description
<code>mq.log.level.ERROR</code>	<code>LogNotification.LOG_LEVEL_ERROR</code>	Error message logged
<code>mq.log.level.WARNING</code>	<code>LogNotification.LOG_LEVEL_WARNING</code>	Warning message logged
<code>mq.log.level.INFO</code>	<code>LogNotification.LOG_LEVEL_INFO</code>	Informational message logged

[Table 3–88](#) shows the methods defined in class `LogNotification` for obtaining details about a log monitor notification.

TABLE 3–88 Data Retrieval Methods for Log Monitor Notifications

Method	Result Type	Description
<code>getLevel</code>	String	Logging level of logged message See Table 3–85 for possible values.
<code>getMessage</code>	String	Body of logged message

Java Virtual Machine

This section describes the MBean used for monitoring the Java Virtual Machine (JVM):

- The JVM monitor MBean monitors the Java Virtual Machine.

The following subsection describes this MBean in detail.

JVM Monitor

Each broker has a single *JVM monitor MBean*, used for monitoring the Java Virtual Machine (JVM).

Note – This MBean is useful only with the Java Development Kit (JDK) version 1.4 or lower. JDK version 1.5 includes built-in MBeans that provide more detailed information on the state of the JVM.

Object Name

The JVM monitor MBean has the following object name:

```
com.sun.messaging.jms.server:type=JVM,subtype=Monitor
```

A string representing this object name is defined as a static constant `JVM_MONITOR_MBEAN_NAME` in the utility class `MQObjectName`.

Attributes

The JVM monitor MBean has the attributes shown in [Table 3–89](#). The names of these attributes are defined as static constants in the utility class `JVMAttributes`.

TABLE 3–89 JVM Monitor Attributes

Name	Type	Settable?	Description
TotalMemory	Long	No	Current total memory, in bytes
InitMemory	Long	No	Initial heap size at JVM startup, in bytes
FreeMemory	Long	No	Amount of memory currently available for use, in bytes
MaxMemory	Long	No	Maximum allowable heap size, in bytes Any memory allocation attempt that would exceed this limit will cause an <code>OutOfMemoryError</code> exception to be thrown.

Alphabetical Reference

Table A-1 is an alphabetical list of Message Queue JMX MBean attributes, with cross-references to the relevant tables in this manual.

TABLE A-1 Alphabetical List of MBean Attributes

Attribute	MBean	Reference
AutoCreateQueueMaxNumActiveConsumers	Destination Manager Configuration	Table 3-44
AutoCreateQueueMaxNumBackupConsumers	Destination Manager Configuration	Table 3-44
AutoCreateQueues	Destination Manager Configuration	Table 3-44
AutoCreateTopics	Destination Manager Configuration	Table 3-44
AvgNumActiveConsumers	Destination Monitor	Table 3-38
AvgNumBackupConsumers	Destination Monitor	Table 3-38
AvgNumConsumers	Destination Monitor	Table 3-38
AvgNumMsgs	Destination Monitor	Table 3-38
AvgTotalMsgBytes	Destination Monitor	Table 3-38
BrokerID	Broker Configuration Broker Monitor	Table 3-1 Table 3-4
ClientID	Connection Monitor	Table 3-24
ClientPlatform	Connection Monitor	Table 3-24
ClusterID	Cluster Configuration Cluster Monitor	Table 3-74 Table 3-78

TABLE A-1 Alphabetical List of MBean Attributes (Continued)

Attribute	MBean	Reference
ConfigFileURL	Cluster Configuration	Table 3-74
	Cluster Monitor	Table 3-78
ConnectionID	Connection Configuration	Table 3-23
	Connection Monitor	Table 3-24
	Destination Monitor	Table 3-38
ConsumerFlowLimit	Destination Configuration	Table 3-32
CreatedByAdmin	Destination Monitor	Table 3-38
DiskReserved	Destination Monitor	Table 3-38
DiskUsed	Destination Monitor	Table 3-38
DiskUtilizationRatio	Destination Monitor	Table 3-38
DMQTruncateBody	Destination Manager Configuration	Table 3-44
Embedded	Broker Monitor	Table 3-4
FreeMemory	JVM Monitor	Table 3-89
HighlyAvailable	Cluster Configuration	Table 3-74
	Cluster Monitor	Table 3-78
Host	Connection Monitor	Table 3-24
InitMemory	JVM Monitor	Table 3-89
InstanceName	Broker Configuration	Table 3-1
	Broker Monitor	Table 3-4
Level	Log Configuration	Table 3-84
LimitBehavior	Destination Configuration	Table 3-32
LocalBrokerInfo	Cluster Configuration	Table 3-74
	Cluster Monitor	Table 3-78
LocalDeliveryPreferred	Destination Configuration	Table 3-32
LocalOnly	Destination Configuration	Table 3-32
LogDeadMsgs	Destination Manager Configuration	Table 3-44
MasterBrokerInfo	Cluster Configuration	Table 3-74
	Cluster Monitor	Table 3-78

TABLE A-1 Alphabetical List of MBean Attributes (Continued)

Attribute	MBean	Reference
MaxBytesPerMsg	Destination Configuration	Table 3-32
	Destination Manager Configuration	Table 3-44
MaxMemory	JVM Monitor	Table 3-89
MaxNumActiveConsumers	Destination Configuration	Table 3-32
MaxNumBackupConsumers	Destination Configuration	Table 3-32
MaxNumMsgs	Destination Configuration	Table 3-32
	Destination Manager Configuration	Table 3-44
MaxNumProducers	Destination Configuration	Table 3-32
MaxThreads	Service Configuration	Table 3-8
	Service Manager Configuration	Table 3-17
MaxTotalMsgBytes	Destination Configuration	Table 3-32
	Destination Manager Configuration	Table 3-44
MinThreads	Service Configuration	Table 3-8
	Service Manager Configuration	Table 3-17
MsgBytesIn	Destination Monitor	Table 3-38
	Service Manager Monitor	Table 3-19
	Service Monitor	Table 3-12
MsgBytesOut	Destination Monitor	Table 3-38
	Service Manager Monitor	Table 3-19
	Service Monitor	Table 3-12
Name	Destination Configuration	Table 3-32
	Destination Monitor	Table 3-38
	Service Configuration	Table 3-8
	Service Monitor	Table 3-12
NumActiveConsumers	Destination Monitor	Table 3-38
NumActiveThreads	Service Manager Monitor	Table 3-19
	Service Monitor	Table 3-12
NumBackupConsumers	Destination Monitor	Table 3-38

TABLE A-1 Alphabetical List of MBean Attributes (Continued)

Attribute	MBean	Reference
NumConnections	Connection Manager Configuration	Table 3-26
	Connection Manager Monitor	Table 3-28
	Service Monitor	Table 3-12
NumConnectionsOpened	Connection Manager Monitor	Table 3-28
	Service Monitor	Table 3-12
NumConnectionsRejected	Connection Manager Monitor	Table 3-28
	Service Monitor	Table 3-12
NumConsumers	Connection Monitor	Table 3-24
	Consumer Manager Configuration	Table 3-59
	Consumer Manager Monitor	Table 3-61
	Destination Monitor	Table 3-38
	Service Monitor	Table 3-12
NumDestinations	Destination Manager Configuration	Table 3-44
	Destination Manager Monitor	Table 3-49
NumMsgs	Destination Manager Monitor	Table 3-49
	Destination Monitor	Table 3-38
NumMsgsHeldInTransaction	Destination Monitor	Table 3-38
NumMsgsIn	Destination Monitor	Table 3-38
	Service Manager Monitor	Table 3-19
	Service Monitor	Table 3-12
NumMsgsInDMQ	Destination Manager Monitor	Table 3-49
NumMsgsOut	Destination Monitor	Table 3-38
	Service Manager Monitor	Table 3-19
	Service Monitor	Table 3-12
NumMsgsPendingAcks	Destination Monitor	Table 3-38
NumPktsIn	Service Manager Monitor	Table 3-19
	Service Monitor	Table 3-12
NumPktsOut	Service Manager Monitor	Table 3-19
	Service Monitor	Table 3-12

TABLE A-1 Alphabetical List of MBean Attributes (Continued)

Attribute	MBean	Reference
NumProducers	Connection Monitor	Table 3-24
	Destination Monitor	Table 3-38
	Producer Manager Configuration	Table 3-53
	Producer Manager Monitor	Table 3-55
	Service Monitor	Table 3-12
NumServices	Service Manager Monitor	Table 3-19
NumTransactions	Transaction Manager Configuration	Table 3-66
	Transaction Manager Monitor	Table 3-68
NumTransactionsCommitted	Transaction Manager Monitor	Table 3-68
NumTransactionsRollback	Transaction Manager Monitor	Table 3-68
PeakMsgBytes	Destination Monitor	Table 3-38
PeakNumActiveConsumers	Destination Monitor	Table 3-38
PeakNumBackupConsumers	Destination Monitor	Table 3-38
PeakNumConsumers	Destination Monitor	Table 3-38
PeakNumMsgs	Destination Monitor	Table 3-38
PeakTotalMsgBytes	Destination Monitor	Table 3-38
PktBytesIn	Service Manager Monitor	Table 3-19
	Service Monitor	Table 3-12
PktBytesOut	Service Manager Monitor	Table 3-19
	Service Monitor	Table 3-12
Port	Broker Configuration	Table 3-1
	Broker Monitor	Table 3-4
	Connection Monitor	Table 3-24
	Service Configuration	Table 3-8
	Service Monitor	Table 3-12
RolloverBytes	Log Configuration	Table 3-84
RolloverSecs	Log Configuration	Table 3-84
ServiceName	Connection Monitor	Table 3-24

TABLE A-1 Alphabetical List of MBean Attributes (Continued)

Attribute	MBean	Reference
State	Destination Monitor	Table 3-38
	Service Monitor	Table 3-12
StateLabel	Destination Monitor	Table 3-38
	Service Monitor	Table 3-12
Temporary	Destination Monitor	Table 3-38
ThreadPoolModel	Service Configuration	Table 3-8
TotalMemory	JVM Monitor	Table 3-89
TotalMsgBytes	Destination Manager Monitor	Table 3-49
	Destination Monitor	Table 3-38
TotalMsgBytesHeldInTransaction	Destination Monitor	Table 3-38
TotalMsgBytesInDMQ	Destination Manager Monitor	Table 3-49
Type	Destination Configuration	Table 3-32
	Destination Monitor	Table 3-38
UseDMQ	Destination Configuration	Table 3-32
User	Connection Monitor	Table 3-24
Version	Broker Configuration	Table 3-1
	Broker Monitor	Table 3-4

Table A-2 is an alphabetical list of Message Queue JMX MBean operations, with cross-references to the relevant tables in this manual.

TABLE A-2 Alphabetical List of MBean Operations

Operation	MBean	Reference
commit	Transaction Manager Configuration	Table 3-67
compact	Destination Configuration	Table 3-35
	Destination Manager Configuration	Table 3-45
create	Destination Manager Configuration	Table 3-45
destroy	Connection Manager Configuration	Table 3-27
	Destination Manager Configuration	Table 3-45

TABLE A-2 Alphabetical List of MBean Operations (Continued)

Operation	MBean	Reference
getActiveConsumerIDs	Destination Monitor	Table 3-41
getBackupConsumerIDs	Destination Monitor	Table 3-41
getBrokerAddresses	Cluster Configuration	Table 3-75
	Cluster Monitor	Table 3-79
getBrokerIDs	Cluster Configuration	Table 3-75
	Cluster Monitor	Table 3-79
getBrokerInfo	Cluster Configuration	Table 3-75
	Cluster Monitor	Table 3-79
getBrokerInfoByAddress	Cluster Configuration	Table 3-75
	Cluster Monitor	Table 3-79
getBrokerInfoByID	Cluster Configuration	Table 3-75
	Cluster Monitor	Table 3-79
getConnection	Destination Monitor	Table 3-41
getConnections	Connection Manager Configuration	Table 3-27
	Connection Manager Monitor	Table 3-29
	Service Monitor	Table 3-14
getConsumerIDs	Connection Monitor	Table 3-25
	Consumer Manager Configuration	Table 3-60
	Consumer Manager Monitor	Table 3-62
	Destination Monitor	Table 3-41
	Service Monitor	Table 3-14
getConsumerInfo	Consumer Manager Monitor	Table 3-62
getConsumerInfoByID	Consumer Manager Monitor	Table 3-62
getDestinations	Destination Manager Configuration	Table 3-45
	Destination Manager Monitor	Table 3-50

TABLE A-2 Alphabetical List of MBean Operations (Continued)

Operation	MBean	Reference
getProducerIDs	Connection Monitor	Table 3-25
	Destination Monitor	Table 3-41
	Producer Manager Configuration	Table 3-54
	Producer Manager Monitor	Table 3-56
	Service Monitor	Table 3-14
getProducerInfo	Producer Manager Monitor	Table 3-56
getProducerInfoByID	Producer Manager Monitor	Table 3-56
getService	Connection Monitor	Table 3-25
getServices	Service Manager Configuration	Table 3-18
	Service Manager Monitor	Table 3-20
getTemporaryDestinations	Connection Monitor	Table 3-25
getTransactionIDs	Transaction Manager Configuration	Table 3-67
	Transaction Manager Monitor	Table 3-69
getTransactionInfo	Transaction Manager Monitor	Table 3-69
getTransactionInfoByID	Transaction Manager Monitor	Table 3-69
pause	Destination Configuration	Table 3-35
	Destination Manager Configuration	Table 3-45
	Service Configuration	Table 3-9
	Service Manager Configuration	Table 3-18
purge	Consumer Manager Configuration	Table 3-60
	Destination Configuration	Table 3-35
quiesce	Broker Configuration	Table 3-2
reload	Cluster Configuration	Table 3-75
resetMetrics	Broker Configuration	Table 3-2
restart	Broker Configuration	Table 3-2

TABLE A-2 Alphabetical List of MBean Operations (Continued)

Operation	MBean	Reference
resume	Destination Configuration	Table 3-35
	Destination Manager Configuration	Table 3-45
	Service Configuration	Table 3-9
	Service Manager Configuration	Table 3-18
rollback	Transaction Manager Configuration	Table 3-67
shutdown	Broker Configuration	Table 3-2
takeover	Broker Configuration	Table 3-2
unquiesce	Broker Configuration	Table 3-2

Table A-3 is an alphabetical list of Message Queue JMX MBean notifications, with cross-references to the relevant tables in this manual.

TABLE A-3 Alphabetical List of MBean Notifications

Notification	MBean	Reference
jmx.attribute.change	Broker Configuration	Table 3-3
	Cluster Configuration	Table 3-77
	Destination Configuration	Table 3-37
	Destination Manager Configuration	Table 3-48
	Log Configuration	Table 3-86
	Service Configuration	Table 3-10
mq.broker.quiesce.complete	Broker Monitor	Table 3-5
mq.broker.quiesce.start	Broker Monitor	Table 3-5
mq.broker.shutdown.start	Broker Monitor	Table 3-5
mq.broker.takeover.complete	Broker Monitor	Table 3-5
	Cluster Monitor	Table 3-82
mq.broker.takeover.fail	Broker Monitor	Table 3-5
	Cluster Monitor	Table 3-82
mq.broker.takeover.start	Broker Monitor	Table 3-5
	Cluster Monitor	Table 3-82

TABLE A-3 Alphabetical List of MBean Notifications (Continued)

Notification	MBean	Reference
mq.cluster.broker.down	Cluster Monitor	Table 3-82
mq.cluster.broker.join	Broker Monitor Cluster Monitor	Table 3-5 Table 3-82
mq.connection.close	Connection Manager Monitor Service Monitor	Table 3-30 Table 3-15
mq.connection.open	Connection Manager Monitor Service Monitor	Table 3-30 Table 3-15
mq.connection.reject	Connection Manager Monitor Service Monitor	Table 3-30 Table 3-15
mq.destination.compact	Destination Manager Monitor Destination Monitor	Table 3-51 Table 3-42
mq.destination.create	Destination Manager Monitor	Table 3-51
mq.destination.destroy	Destination Manager Monitor	Table 3-51
mq.destination.pause	Destination Manager Monitor Destination Monitor	Table 3-51 Table 3-42
mq.destination.purge	Destination Manager Monitor Destination Monitor	Table 3-51 Table 3-42
mq.destination.resume	Destination Manager Monitor Destination Monitor	Table 3-51 Table 3-42
mq.log.level.ERROR	Log Monitor	Table 3-87
mq.log.level.INFO	Log Monitor	Table 3-87
mq.log.level.WARNING	Log Monitor	Table 3-87
mq.service.pause	Service Manager Monitor Service Monitor	Table 3-21 Table 3-15
mq.service.resume	Service Manager Monitor Service Monitor	Table 3-21 Table 3-15
mq.transaction.commit	Transaction Manager Monitor	Table 3-72
mq.transaction.prepare	Transaction Manager Monitor	Table 3-72

TABLE A-3 Alphabetical List of MBean Notifications (Continued)

Notification	MBean	Reference
mq.transaction.rollback	Transaction Manager Monitor	Table 3-72

Index

A

- AcknowledgeMode lookup key
 - message consumer, 86, 87
- AcknowledgeModeLabel lookup key
 - message consumer, 86, 87
- acknowledgment modes, 86
 - table, 87-88
- addNotificationListener method, interface
 - MBeanServerConnection, 44
- Address lookup key
 - broker cluster, 95, 98
- admin connection service name, 25
- AdminConnectionConfiguration utility class, 27, 30
- AdminConnectionFactory class, 27, 30
- administration connection factory
 - configuring, 30
 - defined, 30
 - obtaining JMX connector from, 30-31
 - obtaining JMX connector without, 31-32
- ALL utility constant
 - class DestinationPauseType, 69, 78
- Attribute class, 34
- attribute lists, defined, 34
- AttributeList class, 34
- attributes, MBean
 - accessing, 33-38
 - alphabetical list (table), 105-110
 - broker configuration MBean, 48
 - broker monitor MBean, 51-52
 - cluster configuration MBean, 92-93
 - cluster monitor MBean, 96
 - combining with operations, 40
 - attributes, MBean (*Continued*)
 - connection configuration MBean, 61-62
 - connection manager configuration MBean, 63-64
 - connection manager monitor MBean, 64-65
 - connection monitor MBean, 62-63
 - consumer manager configuration MBean, 84
 - consumer manager monitor MBean, 85
 - defined, 20
 - destination configuration MBean, 66-69
 - destination manager configuration MBean, 75-76
 - destination manager monitor MBean, 78-79
 - destination monitor MBean, 70-73
 - JVM monitor MBean, 103
 - log configuration MBean, 100-101
 - producer manager configuration MBean, 81
 - producer manager monitor MBean, 82
 - service configuration MBean, 54
 - service manager configuration MBean, 58
 - service manager monitor MBean, 59-60
 - service monitor MBean, 55-57
 - transaction manager configuration MBean, 88
 - transaction manager monitor MBean, 89-90
 - authentication credentials, 32
 - AUTO_ACKNOWLEDGE utility constant, interface
 - Session, 87
 - AutoCreateQueueMaxNumActiveConsumers attribute,
 - destination manager configuration MBean, 76
 - AutoCreateQueueMaxNumBackupConsumers attribute,
 - destination manager configuration MBean, 76
 - AutoCreateQueues attribute, destination manager configuration MBean, 75

- AutoCreateTopics attribute, destination manager configuration MBean, 75
- AvgNumActiveConsumers attribute
 - destination monitor MBean, 50,72
- AvgNumBackupConsumers attribute
 - destination monitor MBean, 50,72
- AvgNumConsumers attribute
 - destination monitor MBean, 50,71
- AvgNumMsgs attribute
 - destination monitor MBean, 50,72
- AvgTotalMsgBytes attribute
 - destination monitor MBean, 50,73

- B**
- Broker MBean type, 23
- broker clusters, 92-100
 - cluster identifier, 93,96,100
 - composite data object, lookup keys for (table), 95
 - configuration MBean, 92-95
 - high-availability (HA), 49,93,94,95,96,97,98,99,100
 - monitor MBean, 95-100
- BROKER_CONFIG_MBEAN_NAME utility constant
 - class MQObjectName, 26,48
- broker configuration MBean, 47-51
 - attributes, 48
 - notification, 50-51
 - object name, 48
 - operations, 48-50
- BROKER_DOWN utility constant, class BrokerState, 99
- broker monitor MBean, 51-53
 - attributes, 51-52
 - notification objects, 52
 - notifications, 52-53
 - object name, 51
- BROKER_MONITOR_MBEAN_NAME utility constant
 - class MQObjectName, 26,51
- BROKER_QUIESCE_COMPLETE utility constant, class BrokerNotification, 52
- BROKER_QUIESCE_START utility constant, class BrokerNotification, 52
- BROKER_SHUTDOWN_START utility constant, class BrokerNotification, 52
- BROKER_TAKEOVER_COMPLETE utility constant
 - class BrokerNotification, 52,99
- BROKER_TAKEOVER_FAIL utility constant
 - class BrokerNotification, 52,99
- BROKER_TAKEOVER_START utility constant
 - class BrokerNotification, 52,99
- BrokerAttributes utility class, 28,48,51
- BrokerClusterInfo utility class, 29,95,98
- BrokerID attribute
 - broker configuration MBean, 48
 - broker monitor MBean, 51
- BrokerNotification class, 28
 - BROKER_QUIESCE_COMPLETE constant, 52
 - BROKER_QUIESCE_START constant, 52
 - BROKER_SHUTDOWN_START constant, 52
 - BROKER_TAKEOVER_COMPLETE constant, 52,99
 - BROKER_TAKEOVER_FAIL constant, 52,99
 - BROKER_TAKEOVER_START constant, 52,99
 - data retrieval methods, 52
 - utility constants, 52,99
- BrokerOperations utility class, 28,48
- brokers, *See* message brokers
- BrokerState utility class, 28,98
 - BROKER_DOWN constant, 99
 - OPERATING constant, 98
 - QUIESCE_COMPLETE constant, 99
 - QUIESCE_STARTED constant, 99
 - SHUTDOWN_STARTED constant, 99
 - TAKEOVER_COMPLETE constant, 99
 - TAKEOVER_FAILED constant, 99
 - TAKEOVER_STARTED constant, 99
 - UNKNOWN constant, 99

- C**
- classes
 - AdminConnectionConfiguration, 27,30
 - AdminConnectionFactory, 27,30
 - Attribute, 34
 - AttributeList, 34
 - BrokerAttributes, 28,48,51
 - BrokerClusterInfo, 29,95,98
 - BrokerNotification, 28,52,99
 - BrokerOperations, 28,48

classes (*Continued*)

- BrokerState, 28, 98
- ClusterAttributes, 29, 92, 96
- ClusterNotification, 29, 52, 99
- ClusterOperations, 29, 93, 96
- ConnectionAttributes, 28, 61, 62, 63, 64
- ConnectionNotification, 28, 57, 65
- ConnectionOperations, 28, 63, 64, 65
- ConsumerAttributes, 29, 84, 85
- ConsumerInfo, 29, 86
- ConsumerOperations, 29, 84, 85
- DestinationAttributes, 28, 66, 70, 75, 77, 78
- DestinationLimitBehavior, 29, 68
- DestinationNotification, 28, 74, 79, 80
- DestinationOperations, 28, 69, 73, 76, 79
- DestinationPauseType, 29, 69, 78
- DestinationState, 28, 73
- DestinationType, 24, 28, 68, 73, 77, 83, 87
- JMXConnectorFactory, 31
- JVMAttributes, 29, 103
- LogAttributes, 29, 100
- LogLevel, 29, 101
- LogNotification, 29, 102
- MQNotification, 28
- MQObjectName, 20, 25, 28
- Notification, 44
- NotificationFilterSupport, 45
- ObjectName, 20, 22
- ProducerAttributes, 29, 81, 82
- ProducerInfo, 29, 82
- ProducerOperations, 29, 81, 82
- ServiceAttributes, 28, 54, 55, 58, 59
- ServiceNotification, 28, 57, 58, 60
- ServiceOperations, 28, 54, 57, 58, 60
- ServiceState, 28, 56
- TransactionAttributes, 29, 88, 89
- TransactionInfo, 29, 90
- TransactionNotification, 29, 91, 92
- TransactionOperations, 29, 88, 90
- TransactionState, 29, 91
- utility, 28-29
- CLASSPATH environment variable, 27
- CLIENT_ACKNOWLEDGE utility constant, interface
 - Session, 87
- ClientID attribute, connection monitor MBean, 62
- ClientID lookup key
 - message consumer, 87
 - transaction, 90
- ClientPlatform attribute, connection monitor MBean, 62
- Cluster MBean type, 24
- CLUSTER_BROKER_DOWN utility constant, class
 - ClusterNotification, 99
- CLUSTER_BROKER_JOIN utility constant
 - class ClusterNotification, 52, 99
- CLUSTER_CONFIG_MBEAN_NAME utility constant
 - class MQObjectName, 26, 92
- cluster configuration MBean, 92-95
 - attributes, 92-93
 - notification, 95
 - object name, 92
 - operations, 93-95
- cluster monitor MBean, 95-100
 - attributes, 96
 - notification objects, 99
 - notifications, 99-100
 - object name, 95
 - operations, 96-99
- CLUSTER_MONITOR_MBEAN_NAME utility constant
 - class MQObjectName, 26, 95
- ClusterAttributes utility class, 29, 92, 96
- ClusterID attribute
 - cluster configuration MBean, 93
 - cluster monitor MBean, 96
- ClusterNotification class, 29
 - CLUSTER_BROKER_DOWN constant, 99
 - CLUSTER_BROKER_JOIN constant, 52, 99
 - data retrieval methods, 99
 - utility constants, 52, 99
- ClusterOperations utility class, 29, 93, 96
- clusters, *See* broker clusters
- com.sun.messaging package, 27, 30
- com.sun.messaging.jms.management.server package, 27, 28
- commit operation, transaction manager configuration MBean, 89
- COMMITTED utility constant, class
 - TransactionState, 91

- compact operation
 - destination configuration MBean, 69
 - destination manager configuration MBean, 77
- COMPLETE utility constant, class `TransactionState`, 91
- composite data objects
 - See also* lookup keys
 - for broker clusters, 94, 95, 97
 - defined, 42
 - for message brokers, 93, 96
 - for message consumers, 85, 86
 - for message producers, 82, 83
 - for transactions, 90
- `CompositeData` interface, 42, 82, 85, 86, 90, 93, 94, 95, 96, 97, 98
- `Config` MBean subtype, 24
- `ConfigFileURL` attribute
 - cluster configuration MBean, 93
 - cluster monitor MBean, 96
- configuration MBeans, 21
- connect method
 - class `JMXConnectorFactory`, 31, 32
- `Connection` MBean type, 23
- CONNECTION_CLOSE utility constant
 - class `ConnectionNotification`, 57, 65
- connection configuration MBean, 61-62
 - attribute, 61-62
 - object name, 61
- CONNECTION_MANAGER_CONFIG_MBEAN_NAME utility constant
 - class `MQObjectName`, 26, 63
- connection manager configuration MBean, 63-64
 - attribute, 63-64
 - object name, 63
 - operations, 64
- connection manager monitor MBean, 64-66
 - attributes, 64-65
 - notification objects, 65
 - notifications, 65-66
 - object name, 64
 - operation, 65
- CONNECTION_MANAGER_MONITOR_MBEAN_NAME utility constant
 - class `MQObjectName`, 26, 64
- connection monitor MBean, 62-63
 - connection monitor MBean (*Continued*)
 - attributes, 62-63
 - object name, 62
 - operations, 63
 - CONNECTION_OPEN utility constant
 - class `ConnectionNotification`, 57, 65
 - CONNECTION_REJECT utility constant
 - class `ConnectionNotification`, 57, 65
 - connection services, 53-61
 - configuration MBean, 53-55
 - manager configuration MBean, 58-59
 - manager monitor MBean, 59-61
 - monitor MBean, 55-58
 - names (table), 24-25
 - state values (table), 57
 - `ConnectionAttributes` utility class, 28, 61, 62, 63, 64
 - `ConnectionID` attribute
 - connection configuration MBean, 62
 - connection monitor MBean, 62
 - destination monitor MBean, 71
 - `ConnectionID` lookup key
 - message consumer, 86
 - message producer, 83
 - `ConnectionManager` MBean type, 23
 - `ConnectionNotification` class, 28
 - CONNECTION_CLOSE constant, 57, 65
 - CONNECTION_OPEN constant, 57, 65
 - CONNECTION_REJECT constant, 57, 65
 - data retrieval methods, 65
 - utility constants, 57, 65
 - `ConnectionOperations` utility class, 28, 63, 64, 65
- connections, 61-66
 - configuration MBean, 61-62
 - connection identifier, 61, 62, 64, 71, 83, 86
 - manager configuration MBean, 63-64
 - manager monitor MBean, 64-66
 - monitor MBean, 62-63
- `ConnectionString` lookup key, transaction, 90
- connectors, JMX, *See* JMX connectors
- CONSUMER_MANAGER_CONFIG_MBEAN_NAME utility constant
 - class `MQObjectName`, 26, 84
- consumer manager configuration MBean, 84-85
 - attribute, 84

- consumer manager configuration MBean (*Continued*)
 object name, 84
 operations, 84-85
- consumer manager monitor MBean, 85-88
 attribute, 85
 object name, 85
 operations, 85-88
- CONSUMER_MANAGER_MONITOR_MBEAN_NAME utility
 constant
 class MQObjectName, 26, 85
- ConsumerAttributes utility class, 29, 84, 85
- ConsumerFlowLimit attribute, destination
 configuration MBean, 67
- ConsumerID lookup key, message consumer, 86
- ConsumerInfo utility class, 29, 86
- ConsumerManager MBean type, 24
- ConsumerOperations utility class, 29, 84, 85
- consumers, *See* message consumers
- CONSUMERS utility constant
 class DestinationPauseType, 69, 78
- CONSUMERS_PAUSED utility constant, class
 DestinationState, 73
- create operation
 destination manager configuration MBean, 77
- createConnection method, class
 AdminConnectionFactory, 30
- createConnectionConfig utility method
 class MQObjectName, 26, 61
- createConnectionMonitor utility method
 class MQObjectName, 26, 62
- CREATED utility constant, class TransactionState, 91
- CreatedByAdmin attribute, destination monitor
 MBean, 70
- createDestinationConfig utility method
 class MQObjectName, 26, 66
- createDestinationMonitor utility method
 class MQObjectName, 26, 70
- createServiceConfig utility method
 class MQObjectName, 26, 53
- createServiceMonitor utility method
 class MQObjectName, 26, 55
- CreationTime lookup key, transaction, 90
- CREDENTIALS attribute
 (JMXXConnectorFactory.connect environment), 32
- D**
- dead message queue, 67, 68, 76, 79
- Destination MBean type, 23
- DESTINATION_COMPACT utility constant
 class DestinationNotification, 74, 80
- destination configuration MBean, 66-70
 attributes, 66-69
 notification, 69-70
 object name, 66
 operations, 69
- DESTINATION_CREATE utility constant, class
 DestinationNotification, 79
- DESTINATION_DESTROY utility constant, class
 DestinationNotification, 79
- DESTINATION_MANAGER_CONFIG_MBEAN_NAME utility
 constant
 class MQObjectName, 26, 75
- destination manager configuration MBean, 75-78
 attributes, 75-76
 notification, 78
 object name, 75
 operations, 76-78
- destination manager monitor MBean, 78-80
 attributes, 78-79
 notification objects, 80
 notifications, 79-80
 object name, 78
 operation, 79
- DESTINATION_MANAGER_MONITOR_MBEAN_NAME utility
 constant
 class MQObjectName, 26, 78
- destination monitor MBean, 70-75
 attributes, 70-73
 notification objects, 74
 notifications, 74-75
 object name, 70
 operations, 73-74
- DESTINATION_PAUSE utility constant
 class DestinationNotification, 74, 79
- DESTINATION_PURGE utility constant
 class DestinationNotification, 74, 80
- DESTINATION_RESUME utility constant
 class DestinationNotification, 74, 80
- destination types (table), 24

- DestinationAttributes utility class, 28, 66, 70, 75, 77, 78
 - DestinationLimitBehavior utility class, 29, 68
 - FLOW_CONTROL constant, 68
 - REJECT_NEWEST constant, 69
 - REMOVE_LOW_PRIORITY constant, 68
 - REMOVE_OLDEST constant, 68
 - DestinationManager MBean type, 23
 - DestinationName lookup key
 - message consumer, 86
 - message producer, 83
 - DestinationNotification class, 28
 - data retrieval methods, 74, 80
 - DESTINATION_COMPACT constant, 74, 80
 - DESTINATION_CREATE constant, 79
 - DESTINATION_DESTROY constant, 79
 - DESTINATION_PAUSE constant, 74, 79
 - DESTINATION_PURGE constant, 74, 80
 - DESTINATION_RESUME constant, 74, 80
 - utility constants, 74, 79
 - DestinationOperations utility class, 28, 69, 73, 76, 79
 - DestinationPauseType utility class, 29, 69, 78
 - ALL constant, 69, 78
 - CONSUMERS constant, 69, 78
 - PRODUCERS constant, 69, 78
 - destinations, 66-80
 - configuration MBean, 66-70
 - limit behavior (table), 68-69
 - manager configuration MBean, 75-78
 - manager monitor MBean, 78-80
 - monitor MBean, 70-75
 - pause types (table), 69
 - types (table), 24
 - DestinationState utility class, 28, 73
 - CONSUMERS_PAUSED constant, 73
 - PAUSED constant, 73
 - PRODUCERS_PAUSED constant, 73
 - RUNNING constant, 73
 - UNKNOWN constant, 73
 - DestinationType lookup key
 - message consumer, 86, 87
 - message producer, 83
 - DestinationType utility class, 24, 28, 68, 73, 77, 83, 87
 - destroy operation
 - connection manager configuration MBean, 64
 - destination manager configuration MBean, 77
 - destType property (object name), 23
 - values (table), 24
 - directory variables
 - IMQ_HOME, 16
 - IMQ_JAVAHOME, 16
 - IMQ_VARHOME, 16
 - DiskReserved attribute, destination monitor MBean, 73
 - DiskUsed attribute, destination monitor MBean, 73
 - DiskUtilizationRatio attribute, destination monitor MBean, 73
 - distributed transaction identifier (XID), 90
 - DMQTruncateBody attribute, destination manager configuration MBean, 76
 - DUPS_OK_ACKNOWLEDGE utility constant, interface Session, 87
 - Durable lookup key, message consumer, 86
 - DurableActive lookup key, message consumer, 87
 - DurableName lookup key, message consumer, 86
- ## E
- Embedded attribute, broker monitor MBean, 52
 - enableType method, class NotificationFilterSupport, 45
 - ERROR utility constant, class LogLevel, 101
 - example applications, 14
- ## F
- FAILED utility constant, class TransactionState, 91
 - FLOW_CONTROL utility constant, class DestinationLimitBehavior, 68
 - FlowPaused lookup key
 - message consumer, 87
 - message producer, 83
 - FreeMemory attribute, JVM monitor MBean, 103

G

- getActiveConsumerIDs operation, destination monitor MBean, 74
- getAttribute method, interface MBeanServerConnection, 33
- getAttributes method, interface MBeanServerConnection, 34
- getBackupConsumerIDs operation, destination monitor MBean, 74
- getBrokerAddress method
 - class BrokerNotification, 52
 - class ClusterNotification, 100
- getBrokerAddresses operation
 - cluster configuration MBean, 94
 - cluster monitor MBean, 97
- getBrokerID method
 - class BrokerNotification, 52
 - class ClusterNotification, 100
- getBrokerIDs operation
 - cluster configuration MBean, 94
 - cluster monitor MBean, 97
- getBrokerInfo operation
 - cluster configuration MBean, 94, 95
 - cluster monitor MBean, 97, 98
- getBrokerInfoByAddress operation
 - cluster configuration MBean, 94, 95
 - cluster monitor MBean, 97, 98
- getBrokerInfoByID operation
 - cluster configuration MBean, 94, 95
 - cluster monitor MBean, 97, 98
- getClusterID method, class ClusterNotification, 100
- getConnection operation, destination monitor MBean, 74
- getConnectionID method, class ConnectionNotification, 65
- getConnections operation
 - connection manager configuration MBean, 64
 - connection manager monitor MBean, 65
 - service monitor MBean, 57
- getConsumerIDs operation
 - connection monitor MBean, 63
 - consumer manager configuration MBean, 84
 - consumer manager monitor MBean, 85
- getConsumerIDs operation (*Continued*)
 - destination monitor MBean, 74
 - service monitor MBean, 57
- getConsumerInfo operation
 - consumer manager monitor MBean, 86
- getConsumerInfoByID operation
 - consumer manager monitor MBean, 85, 86
- getCreatedByAdmin method
 - class DestinationNotification, 75, 80
- getDestinationName method
 - class DestinationNotification, 75, 80
- getDestinations operation
 - destination manager configuration MBean, 77
 - destination manager monitor MBean, 79
- getDestinationType method
 - class DestinationNotification, 75, 80
- getFailedBrokerID method, class BrokerNotification, 53
- getLevel method, class LogNotification, 102
- getMBeanServerConnection method, class JMXConnector, 30
- getMessage method, class LogNotification, 102
- getName method, class Attribute, 34
- getPauseType method
 - class DestinationNotification, 75, 80
- getProducerIDs operation
 - connection monitor MBean, 63
 - destination monitor MBean, 74
 - producer manager configuration MBean, 81
 - producer manager monitor MBean, 82
 - service monitor MBean, 57
- getProducerInfo operation
 - producer manager monitor MBean, 82
- getProducerInfoByID operation
 - producer manager monitor MBean, 82
- getRemoteHost method, class ConnectionNotification, 65
- getService operation, connection monitor MBean, 63
- getServiceName method
 - class ConnectionNotification, 65
 - class ServiceNotification, 58, 61
- getServices operation
 - service manager configuration MBean, 59
 - service manager monitor MBean, 60

getTemporaryDestinations operation, connection monitor MBean, 63
 getTransactionID method, class TransactionNotification, 92
 getTransactionIDs operation
 transaction manager configuration MBean, 89
 transaction manager monitor MBean, 90
 getTransactionInfo operation
 transaction manager monitor MBean, 90
 getTransactionInfoByID operation
 transaction manager monitor MBean, 90
 getUsername method, class ConnectionNotification, 66
 getValue method, class Attribute, 34

H

HA, *See* high-availability broker clusters
 handback objects, 44
 handleNotification method, interface NotificationListener, 44
 high-availability (HA) broker clusters
 ClusterID attribute, cluster configuration MBean, 93
 ClusterID attribute, cluster monitor MBean, 96
 getBrokerAddresses operation, cluster configuration MBean, 94
 getBrokerAddresses operation, cluster monitor MBean, 97
 getBrokerIDs operation, cluster configuration MBean, 94
 getBrokerIDs operation, cluster monitor MBean, 97
 getBrokerInfo operation, cluster configuration MBean, 94
 getBrokerInfo operation, cluster monitor MBean, 97
 HighlyAvailable attribute, cluster configuration MBean, 93
 HighlyAvailable attribute, cluster monitor MBean, 96
 ID lookup key, composite data object, 95, 98
 isHighlyAvailable method, class ClusterNotification, 100

high-availability (HA) broker clusters (*Continued*)
 mq.broker.takeover.complete notification, cluster monitor MBean, 99
 mq.broker.takeover.fail notification, cluster monitor MBean, 99
 mq.broker.takeover.start notification, cluster monitor MBean, 99
 NumMsgs lookup key, composite data object, 98
 shutdown operation, broker configuration MBean, 49
 StatusTimestamp lookup key, composite data object, 98
 TAKEOVER_COMPLETE state, 99
 TAKEOVER_FAILED state, 99
 TAKEOVER_STARTED state, 99
 TakeoverBrokerID lookup key, composite data object, 98
 HighlyAvailable attribute
 cluster configuration MBean, 93
 cluster monitor MBean, 96
 Host attribute, connection monitor MBean, 62
 Host lookup key
 message consumer, 86
 message producer, 83
 HTTP, *See* Hypertext Transfer Protocol
 httpjms connection service name, 24
 HTTPS, *See* Hypertext Transfer Protocol, Secure
 httpsjms connection service name, 25
 Hypertext Transfer Protocol (HTTP), 24
 Hypertext Transfer Protocol, Secure (HTTPS), 25

I

ID lookup key
 broker cluster, 95, 98
 id property (object name), 23
 IMQ_HOME directory variable, 16
 IMQ_JAVAHOME directory variable, 16
 IMQ_VARHOME directory variable, 16
 imqjmx.jar file, 27
 INCOMPLETE utility constant, class TransactionState, 91
 INFO utility constant, class LogLevel, 101
 InitMemory attribute, JVM monitor MBean, 103

InstanceName attribute
 broker configuration MBean, 48
 broker monitor MBean, 51
 interfaces
 CompositeData, 42, 82, 85, 86, 90, 93, 94, 95, 96, 97, 98
 NotificationListener, 44
 Session, 87
 invoke method
 interface MBeanServerConnection, 38, 39
 isHighlyAvailable method, class
 ClusterNotification, 100
 isMasterBroker method, class
 ClusterNotification, 100

J

Java Management Extensions (JMX) Specification, 11, 13
Java Management Extensions (JMX) Technology Overview, 13
Java Management Extensions (JMX) Technology Tutorial, 13
 Java Monitoring and Management Console
 (jconsole), 56, 71, 86, 91, 98
 Java Naming and Directory Interface (JNDI), 32
 JavaDoc documentation, 13-14
 jconsole, *See* Java Monitoring and Management Console
 jms connection service name, 24
 jmx.attribute.change notification
 broker configuration MBean, 51
 cluster configuration MBean, 95
 destination configuration MBean, 70
 destination manager configuration MBean, 78
 log configuration MBean, 101
 service configuration MBean, 55
 JMX connectors
 defined, 20
 obtaining from administration connection factory, 30-31
 obtaining without administration connection factory, 31-32

JMX service URLs
 parameter to JMXConnectorFactory.connect method, 32
 syntax, 32-33
 JMXConnectorFactory class, 31
 JNDI, *See* Java Naming and Directory Interface
 JVM, *See* Java Virtual Machine
 JVM (Java Virtual Machine), 103
 monitor MBean, 103
 JVM MBean type, 24
 JVM monitor MBean, 21, 103
 attributes, 103
 object name, 103
 JVM_MONITOR_MBEAN_NAME utility constant
 class MQObjectName, 26, 103
 JVMAttributes utility class, 29, 103

L

LastAckTime lookup key, message consumer, 87
 Level attribute
 log configuration MBean, 101
 limit behavior, destinations, 68-69
 LimitBehavior attribute
 destination configuration MBean, 67, 68
 LocalBrokerInfo attribute
 cluster configuration MBean, 93, 95
 cluster monitor MBean, 96, 98
 LocalDeliveryPreferred attribute, destination configuration MBean, 68
 LocalOnly attribute, destination configuration MBean, 68
 Log MBean type, 24
 LOG_CONFIG_MBEAN_NAME utility constant
 class MQObjectName, 26, 100
 log configuration MBean, 100-101
 attributes, 100-101
 notification, 101
 object name, 100
 LOG_LEVEL_ERROR utility constant, class LogNotification, 102
 LOG_LEVEL_INFO utility constant, class LogNotification, 102

- LOG_LEVEL_WARNING utility constant, class
 - LogNotification, 102
- log monitor MBean, 101-102
 - notification objects, 102
 - notifications, 102
 - object name, 102
- LOG_MONITOR_MBEAN_NAME utility constant
 - class MQObjectName, 26, 102
- LogAttributes utility class, 29, 100
- LogDeadMsgs attribute, destination manager configuration MBean, 76
- logging, 100-102
 - configuration MBean, 100-101
 - monitor MBean, 101-102
- LogLevel utility class, 29, 101
 - ERROR constant, 101
 - INFO constant, 101
 - NONE constant, 101
 - UNKNOWN constant, 101
 - WARNING constant, 101
- LogNotification class, 29
 - data retrieval methods, 102
 - LOG_LEVEL_ERROR constant, 102
 - LOG_LEVEL_INFO constant, 102
 - LOG_LEVEL_WARNING constant, 102
 - utility constants, 102
- lookup keys
 - for broker clusters, 95
 - defined, 42
 - for message consumers, 86-87
 - for message producers, 83
 - for transactions, 90-91
- M**
- managed beans, *See* MBeans
- manager MBeans, 21-22
- MasterBrokerInfo attribute
 - cluster configuration MBean, 93, 95
 - cluster monitor MBean, 96, 98
- MaxBytesPerMsg attribute
 - destination configuration MBean, 67
 - destination manager configuration MBean, 76
- MaxMemory attribute, JVM monitor MBean, 103
- MaxNumActiveConsumers attribute, destination configuration MBean, 67
- MaxNumBackupConsumers attribute, destination configuration MBean, 67
- MaxNumMsgs attribute
 - destination configuration MBean, 67
 - destination manager configuration MBean, 75
- MaxNumProducers attribute, destination configuration MBean, 67
- MaxThreads attribute
 - service configuration MBean, 54
 - service manager configuration MBean, 58
- MaxTotalMsgBytes attribute
 - destination configuration MBean, 67
 - destination manager configuration MBean, 76
- MBean server
 - connecting to, 29-33
 - connection, defined, 29
 - defined, 20
- MBeans
 - attributes, accessing, 33-38
 - broker configuration, 47-51
 - broker monitor, 51-53
 - cluster configuration, 92-95
 - cluster monitor, 95-100
 - combining operations and attributes, 40
 - configuration, defined, 21
 - connection configuration, 61-62
 - connection manager configuration, 63-64
 - connection manager monitor, 64-66
 - connection monitor, 62-63
 - consumer manager configuration, 84-85
 - consumer manager monitor, 85-88
 - defined, 19
 - destination configuration, 66-70
 - destination manager configuration, 75-78
 - destination manager monitor, 78-80
 - destination monitor, 70-75
 - JVM monitor, 21, 103
 - log configuration, 100-101
 - log monitor, 101-102
 - manager, defined, 21-22
 - monitor, defined, 21
 - notifications, receiving, 44-46

MBeans (*Continued*)

- operations, invoking, 38-44
 - producer manager configuration, 80-81
 - producer manager monitor, 81-83
 - resource, defined, 20-21
 - server
 - See MBean server
 - service configuration, 53-55
 - service manager configuration, 58-59
 - service manager monitor, 59-61
 - service monitor, 55-58
 - subtypes (table), 24
 - transaction manager configuration, 88-89
 - transaction manager monitor, 89-92
 - types (table), 23-24
 - using, 33-46
- message brokers, 47-53
- broker identifier, 48, 49, 51, 52, 53, 94, 95, 97, 98, 100
 - configuration MBean, 47-51
 - monitor MBean, 51-53
 - state values (table), 98-99
- message consumers, 83-88
- acknowledgment mode, 86, 87-88
 - composite data object, lookup keys for (table), 86-87
 - consumer identifier, 57, 63, 74, 84, 85, 86
 - manager configuration MBean, 84-85
 - manager monitor MBean, 85-88
- message producers, 80-83
- composite data object, lookup keys for (table), 83
 - manager configuration MBean, 80-81
 - manager monitor MBean, 81-83
 - producer identifier, 57, 63, 74, 81, 82, 83
- message string (notifications), 44
- methods
- addNotificationListener (interface MBeanServerConnection), 44
 - connect (class JMXConnectorFactory), 31
 - createConnection (class AdminConnectionFactory), 30
 - createConnectionConfig (class MQObjectName), 26, 61

methods (*Continued*)

- createConnectionMonitor (class MQObjectName), 26, 62
- createDestinationConfig (class MQObjectName), 26, 66
- createDestinationMonitor (class MQObjectName), 26, 70
- createServiceConfig (class MQObjectName), 26, 53
- createServiceMonitor (class MQObjectName), 26, 55
- enableType (class NotificationFilterSupport), 45
- getAttribute (interface MBeanServerConnection), 33
- getAttributes (interface MBeanServerConnection), 34
- getBrokerAddress (class BrokerNotification), 52
- getBrokerAddress (class ClusterNotification), 100
- getBrokerID (class BrokerNotification), 52
- getBrokerID (class ClusterNotification), 100
- getClusterID (class ClusterNotification), 100
- getConnectionID (class ConnectionNotification), 65
- getCreatedByAdmin (class DestinationNotification), 75, 80
- getDestinationName (class DestinationNotification), 75, 80
- getDestinationType (class DestinationNotification), 75, 80
- getFailedBrokerID (class BrokerNotification), 53
- getLevel (class LogNotification), 102
- getMBeanServerConnection (class JMXConnector), 30
- getMessage (class LogNotification), 102
- getName (class Attribute), 34
- getPauseType (class DestinationNotification), 75, 80
- getRemoteHost (class ConnectionNotification), 65
- getServiceName (class ConnectionNotification), 65

methods (*Continued*)

- getServiceName (class ServiceNotification), 58, 61
- getTransactionID (class TransactionNotification), 92
- getUserName (class ConnectionNotification), 66
- getValue (class Attribute), 34
- handleNotification (interface NotificationListener), 44
- invoke (interface MBeanServerConnection), 38, 39
- isHighlyAvailable (class ClusterNotification), 100
- isMasterBroker (class ClusterNotification), 100
- setAttribute (interface MBeanServerConnection), 35
- setAttributes (interface MBeanServerConnection), 36
- setProperty (class AdminConnectionFactory), 30
- MinThreads attribute
 - service configuration MBean, 54
 - service manager configuration MBean, 58
- Monitor MBean subtype, 24
- monitor MBeans, 21
- mq.broker.quiesce.complete notification, broker monitor MBean, 52
- mq.broker.quiesce.start notification, broker monitor MBean, 52
- mq.broker.shutdown.start notification, broker monitor MBean, 52
- mq.broker.takeover.complete notification
 - broker monitor MBean, 52
 - cluster monitor MBean, 99
- mq.broker.takeover.fail notification
 - broker monitor MBean, 52
 - cluster monitor MBean, 99
- mq.broker.takeover.start notification
 - broker monitor MBean, 52
 - cluster monitor MBean, 99
- mq.cluster.broker.down notification, cluster monitor MBean, 99
- mq.cluster.broker.join notification
 - broker monitor MBean, 52
 - cluster monitor MBean, 99
- mq.connection.close notification
 - connection manager monitor MBean, 65
 - service monitor MBean, 57
- mq.connection.open notification
 - connection manager monitor MBean, 65
 - service monitor MBean, 57
- mq.connection.reject notification
 - connection manager monitor MBean, 65
 - service monitor MBean, 57
- mq.destination.compact notification
 - destination manager monitor MBean, 80
 - destination monitor MBean, 74
- mq.destination.create notification, destination manager monitor MBean, 79
- mq.destination.destroy notification, destination manager monitor MBean, 79
- mq.destination.pause notification
 - destination manager monitor MBean, 79
 - destination monitor MBean, 74
- mq.destination.purge notification
 - destination manager monitor MBean, 80
 - destination monitor MBean, 74
- mq.destination.resume notification
 - destination manager monitor MBean, 80
 - destination monitor MBean, 74
- mq.log.level.ERROR notification, log monitor MBean, 102
- mq.log.level.INFO notification, log monitor MBean, 102
- mq.log.level.WARNING notification, log monitor MBean, 102
- mq.service.pause notification
 - service manager monitor MBean, 60
 - service monitor MBean, 57
- mq.service.resume notification
 - service manager monitor MBean, 60
 - service monitor MBean, 57
- mq.transaction.commit notification, transaction manager monitor MBean, 92
- mq.transaction.prepare notification, transaction manager monitor MBean, 92
- mq.transaction.rollback notification, transaction manager monitor MBean, 92
- MQNotification class, 28

MQObjectName utility class, 20, 25, 28

- BROKER_CONFIG_MBEAN_NAME constant, 26, 48
- BROKER_MONITOR_MBEAN_NAME constant, 26, 51
- CLUSTER_CONFIG_MBEAN_NAME constant, 26, 92
- CLUSTER_MONITOR_MBEAN_NAME constant, 26, 95
- CONNECTION_MANAGER_CONFIG_MBEAN_NAME constant, 26, 63
- CONNECTION_MANAGER_MONITOR_MBEAN_NAME constant, 26, 64
- CONSUMER_MANAGER_CONFIG_MBEAN_NAME constant, 26, 84
- CONSUMER_MANAGER_MONITOR_MBEAN_NAME constant, 26, 85
- createConnectionConfig method, 26, 61
- createConnectionMonitor method, 26, 62
- createDestinationConfig method, 26, 66
- createDestinationMonitor method, 26, 70
- createServiceConfig method, 26, 53
- createServiceMonitor method, 26, 55
- DESTINATION_MANAGER_CONFIG_MBEAN_NAME constant, 26, 75
- DESTINATION_MANAGER_MONITOR_MBEAN_NAME constant, 26, 78
- JVM_MONITOR_MBEAN_NAME constant, 26, 103
- LOG_CONFIG_MBEAN_NAME constant, 26, 100
- LOG_MONITOR_MBEAN_NAME constant, 26, 102
- PRODUCER_MANAGER_CONFIG_MBEAN_NAME constant, 26, 81
- PRODUCER_MANAGER_MONITOR_MBEAN_NAME constant, 26, 81
- SERVICE_MANAGER_CONFIG_MBEAN_NAME constant, 26, 58
- SERVICE_MANAGER_MONITOR_MBEAN_NAME constant, 26, 59
- TRANSACTION_MANAGER_CONFIG_MBEAN_NAME constant, 26, 88
- TRANSACTION_MANAGER_MONITOR_MBEAN_NAME constant, 26, 89

MsgBytesIn attribute

- destination monitor MBean, 50, 72
- service manager monitor MBean, 50, 59
- service monitor MBean, 49, 56

MsgBytesOut attribute

- destination monitor MBean, 50, 72

MsgBytesOut attribute (*Continued*)

- service manager monitor MBean, 50, 59
- service monitor MBean, 49, 56

N

Name attribute

- destination configuration MBean, 67, 77
- destination monitor MBean, 70
- service configuration MBean, 54
- service monitor MBean, 56

name property (object name), 23

- values (table), 24-25

NO_ACKNOWLEDGE utility constant, interface Session, 87

NONE utility constant, class LogLevel, 101

Notification class, 44

notification filters, 44

notification listeners

- defined, 44
- example, 45
- for log notifications, 102
- registering, 45-46

notification objects

- for broker notifications, 52
- for cluster notifications, 99
- for connection notifications, 65
- for connection service notifications, 60
- defined, 44
- for destination notifications, 74, 80
- for log notifications, 102
- for service notifications, 58
- for transaction notifications, 92

NotificationFilterSupport class, 45

NotificationListener interface, 44

notifications, MBean

- alphabetical list (table), 113-115
- broker configuration MBean, 50-51
- broker monitor MBean, 52-53
- cluster configuration MBean, 95
- cluster monitor MBean, 99-100
- connection manager monitor MBean, 65-66
- defined, 20
- destination configuration MBean, 69-70
- destination manager configuration MBean, 78

notifications, MBean (*Continued*)

- destination manager monitor MBean, 79-80
 - destination monitor MBean, 74-75
 - log configuration MBean, 101
 - log monitor MBean, 102
 - receiving, 44-46
 - service configuration MBean, 55
 - service manager monitor MBean, 60-61
 - service monitor MBean, 57-58
 - transaction manager monitor MBean, 91-92
- NumAcks lookup key, transaction, 91
- NumActiveConsumers attribute, destination monitor MBean, 71
- NumActiveThreads attribute
- service manager monitor MBean, 59
 - service monitor MBean, 56
- NumBackupConsumers attribute, destination monitor MBean, 72
- NumConnections attribute
- connection manager configuration MBean, 64
 - connection manager monitor MBean, 65
 - service monitor MBean, 56
- NumConnectionsOpened attribute
- connection manager monitor MBean, 50, 65
 - service monitor MBean, 49, 56
- NumConnectionsRejected attribute
- connection manager monitor MBean, 50, 65
 - service monitor MBean, 49, 56
- NumConsumers attribute
- connection monitor MBean, 63
 - consumer manager configuration MBean, 84
 - consumer manager monitor MBean, 85
 - destination monitor MBean, 71
 - service monitor MBean, 56
- NumDestinations attribute
- destination manager configuration MBean, 75
 - destination manager monitor MBean, 79
- NumMsgs attribute
- destination manager monitor MBean, 79
 - destination monitor MBean, 72
- NumMsgs lookup key
- broker cluster, 98
 - message consumer, 87
 - message producer, 83

NumMsgs lookup key (*Continued*)

- transaction, 91
- NumMsgsHeldInTransaction attribute, destination monitor MBean, 72
- NumMsgsIn attribute
- destination monitor MBean, 50, 72
 - service manager monitor MBean, 50, 59
 - service monitor MBean, 49, 56
- NumMsgsInDMQ attribute, destination manager monitor MBean, 79
- NumMsgsOut attribute
- destination monitor MBean, 50, 72
 - service manager monitor MBean, 50, 59
 - service monitor MBean, 49, 56
- NumMsgsPendingAcks attribute, destination monitor MBean, 72
- NumMsgsPendingAcks lookup key, message consumer, 87
- NumPktsIn attribute
- service manager monitor MBean, 50, 60
 - service monitor MBean, 49, 56
- NumPktsOut attribute
- service manager monitor MBean, 50, 60
 - service monitor MBean, 49, 56
- NumProducers attribute
- connection monitor MBean, 63
 - destination monitor MBean, 71
 - producer manager configuration MBean, 81
 - producer manager monitor MBean, 82
 - service monitor MBean, 56
- NumServices attribute, service manager monitor MBean, 59
- NumTransactions attribute
- transaction manager configuration MBean, 88
 - transaction manager monitor MBean, 89
- NumTransactionsCommitted attribute
- transaction manager monitor MBean, 50, 89
- NumTransactionsRollback attribute
- transaction manager monitor MBean, 50, 89

O

- object names, 22-26
 - broker configuration MBean, 48

object names (*Continued*)

- broker monitor MBean, 51
- cluster configuration MBean, 92
- cluster monitor MBean, 95
- connection configuration MBean, 61
- connection manager configuration MBean, 63
- connection manager monitor MBean, 64
- connection monitor MBean, 62
- consumer manager configuration MBean, 84
- consumer manager monitor MBean, 85
- defined, 20
- destination configuration MBean, 66
- destination manager configuration MBean, 75
- destination manager monitor MBean, 78
- destination monitor MBean, 70
- desttype values (table), 24
- examples, 25
- JVM monitor MBean, 103
- log configuration MBean, 100
- log monitor MBean, 102
- name values (table), 24-25
- producer manager configuration MBean, 81
- producer manager monitor MBean, 81
- properties (table), 23
- service configuration MBean, 53-54
- service manager configuration MBean, 58
- service manager monitor MBean, 59
- service monitor MBean, 55
- subtype values (table), 24
- syntax, 22
- transaction manager configuration MBean, 88
- transaction manager monitor MBean, 89
- type values (table), 23-24
- utility constants and methods (table), 26

ObjectName class, 20, 22

OPERATING utility constant, class BrokerState, 98

operations, MBean

- alphabetical list (table), 110-113
- broker configuration MBean, 48-50
- cluster configuration MBean, 93-95
- cluster monitor MBean, 96-99
- combining with attributes, 40
- connection manager configuration MBean, 64
- connection manager monitor MBean, 65

operations, MBean (*Continued*)

- connection monitor MBean, 63
- consumer manager configuration MBean, 84-85
- consumer manager monitor MBean, 85-88
- defined, 20
- destination configuration MBean, 69
- destination manager configuration MBean, 76-78
- destination manager monitor MBean, 79
- destination monitor MBean, 73-74
- invoking, 38-44
- producer manager configuration MBean, 81
- producer manager monitor MBean, 82-83
- service configuration MBean, 54-55
- service manager configuration MBean, 58-59
- service manager monitor MBean, 60
- service monitor MBean, 57
- transaction manager configuration MBean, 88-89
- transaction manager monitor MBean, 90-91

P

packages

- com.sun.messaging, 27, 30
- com.sun.messaging.jms.management.server, 27, 28

pause operation

- destination configuration MBean, 69
- destination manager configuration MBean, 77, 78
- service configuration MBean, 54
- service manager configuration MBean, 59

pause types, destination, 69

PAUSED utility constant

- class DestinationState, 73
- class ServiceState, 57

PeakMsgBytes attribute

- destination monitor MBean, 50, 72

PeakNumActiveConsumers attribute

- destination monitor MBean, 50, 71

PeakNumBackupConsumers attribute

- destination monitor MBean, 50, 72

PeakNumConsumers attribute

- destination monitor MBean, 50, 71

PeakNumMsgs attribute

- destination monitor MBean, 50, 72

- PeakTotalMsgBytes attribute
 - destination monitor MBean, 50, 73
- PktBytesIn attribute
 - service manager monitor MBean, 50, 60
 - service monitor MBean, 49, 56
- PktBytesOut attribute
 - service manager monitor MBean, 50, 60
 - service monitor MBean, 49, 56
- Port attribute
 - broker configuration MBean, 48
 - broker monitor MBean, 51
 - connection monitor MBean, 62
 - service configuration MBean, 54
 - service monitor MBean, 54, 56
- PREPARED utility constant, class TransactionState, 91
- PRODUCER_MANAGER_CONFIG_MBEAN_NAME utility constant
 - class MQObjectName, 26, 81
- producer manager configuration MBean, 80-81
 - attribute, 81
 - object name, 81
 - operation, 81
- producer manager monitor MBean, 81-83
 - attribute, 82
 - object name, 81
 - operations, 82-83
- PRODUCER_MANAGER_MONITOR_MBEAN_NAME utility constant
 - class MQObjectName, 26, 81
- ProducerAttributes utility class, 29, 81, 82
- ProducerID lookup key, message producer, 83
- ProducerInfo utility class, 29, 82
- ProducerManager MBean type, 23
- ProducerOperations utility class, 29, 81, 82
- producers, *See* message producers
- PRODUCERS utility constant
 - class DestinationPauseType, 69, 78
- PRODUCERS_PAUSED utility constant, class DestinationState, 73
- protocol types, 25
- purge operation
 - consumer manager configuration MBean, 84
 - destination configuration MBean, 69
- Q**
 - q destination type, 24, 68, 73, 78, 83, 87
 - QUEUE utility constant
 - class DestinationType, 24, 68, 73, 78, 83, 87
 - quiesce operation, broker configuration MBean, 49
 - QUIESCE_COMPLETE utility constant, class BrokerState, 99
 - QUIESCE_STARTED utility constant, class BrokerState, 99
 - QUIESCED utility constant, class ServiceState, 57
- R**
 - REJECT_NEWEST utility constant, class DestinationLimitBehavior, 69
 - reload operation, cluster configuration MBean, 95
 - remote method invocation (RMI), 29
 - JMX service URL syntax, 32
 - registry, 32
 - stub, 32
 - REMOVE_LOW_PRIORITY utility constant, class DestinationLimitBehavior, 68
 - REMOVE_OLDEST utility constant, class DestinationLimitBehavior, 68
 - resetMetrics operation, broker configuration MBean, 49
 - resource MBeans, 20-21
 - restart operation, broker configuration MBean, 49
 - resume operation
 - destination configuration MBean, 69
 - destination manager configuration MBean, 77
 - service configuration MBean, 54
 - service manager configuration MBean, 59
 - RMI, *See* remote method invocation
 - rollback operation, transaction manager configuration MBean, 89
 - ROLLEDBACK utility constant, class TransactionState, 91
 - RolloverBytes attribute, log configuration MBean, 101
 - RolloverSecs attribute, log configuration MBean, 101
 - RUNNING utility constant
 - class DestinationState, 73
 - class ServiceState, 57

S

- Secure Hypertext Transfer Protocol (HTTPS), 25
- Secure Socket Layer (SSL), 24, 25
- Selector lookup key, message consumer, 86
- server, MBean, *See* MBean server
- Service MBean type, 23
- service configuration MBean, 53-55
 - attributes, 54
 - notification, 55
 - object name, 53-54
 - operations, 54-55
- SERVICE_MANAGER_CONFIG_MBEAN_NAME utility constant
 - class MQObjectName, 26, 58
- service manager configuration MBean, 58-59
 - attributes, 58
 - object name, 58
 - operations, 58-59
- service manager monitor MBean, 59-61
 - attributes, 59-60
 - notification objects, 60
 - notifications, 60-61
 - object name, 59
 - operation, 60
- SERVICE_MANAGER_MONITOR_MBEAN_NAME utility constant
 - class MQObjectName, 26, 59
- service monitor MBean, 55-58
 - attributes, 55-57
 - notification objects, 58
 - notifications, 57-58
 - object name, 55
 - operations, 57
- SERVICE_PAUSE utility constant
 - class ServiceNotification, 57, 60
- SERVICE_RESUME utility constant
 - class ServiceNotification, 57, 60
- service URLs, JMX
 - parameter to `JMXConnectorFactory.connect` method, 32
 - syntax, 32-33
- ServiceAttributes utility class, 28, 54, 55, 58, 59
- ServiceManager MBean type, 23
- ServiceName attribute, connection monitor MBean, 62
- ServiceName lookup key
 - message consumer, 86
 - message producer, 83
- ServiceNotification class, 28
 - data retrieval method, 58, 60
 - SERVICE_PAUSE constant, 57, 60
 - SERVICE_RESUME constant, 57, 60
 - utility constants, 57, 60
- ServiceOperations utility class, 28, 54, 57, 58, 60
- services, *See* connection services
- ServiceState utility class, 28, 56
 - PAUSED constant, 57
 - QUIESCED constant, 57
 - RUNNING constant, 57
 - UNKNOWN constant, 57
- Session interface, 87
 - AUTO_ACKNOWLEDGE constant, 87
 - CLIENT_ACKNOWLEDGE constant, 87
 - DUPS_OK_ACKNOWLEDGE constant, 87
 - NO_ACKNOWLEDGE constant, 87
 - SESSION_TRANSACTED constant, 87
- SESSION_TRANSACTED utility constant, interface
 - Session, 87
- setAttribute method, interface
 - MBeanServerConnection, 35
- setAttributes method, interface
 - MBeanServerConnection, 36
- setProperty method, class
 - AdminConnectionFactory, 30
- shutdown operation
 - broker configuration MBean, 49
- SHUTDOWN_STARTED utility constant, class
 - BrokerState, 99
- SSL, *See* Secure Socket Layer
- ssladmin connection service name, 25
- ssljms connection service name, 24
- STARTED utility constant, class TransactionState, 91
- State attribute
 - destination monitor MBean, 71, 73
 - service monitor MBean, 56
- State lookup key
 - broker cluster, 98
 - transaction, 91

- StateLabel attribute
 - destination monitor MBean, 71,73
 - service monitor MBean, 56
 - StateLabel lookup key
 - broker cluster, 98
 - transaction, 91
 - StatusTimestamp lookup key, broker cluster, 98
 - subtype property (object name), 23
 - values (table), 24
- T**
- t destination type, 24, 68, 73, 78, 83, 87
 - takeover operation, broker configuration MBean, 49
 - TAKEOVER_COMPLETE utility constant, class
 - BrokerState, 99
 - TAKEOVER_FAILED utility constant, class
 - BrokerState, 99
 - TAKEOVER_STARTED utility constant, class
 - BrokerState, 99
 - TakeoverBrokerID lookup key, broker cluster, 98
 - TCP, *See* Transmission Control Protocol
 - Temporary attribute, destination monitor MBean, 70
 - ThreadPoolModel attribute, service configuration MBean, 54
 - TIMED_OUT utility constant, class
 - TransactionState, 91
 - TLS, *See* Transport Layer Security
 - TOPIC utility constant
 - class DestinationType, 24, 68, 73, 78, 83, 87
 - TotalMemory attribute, JVM monitor MBean, 103
 - TotalMsgBytes attribute
 - destination manager monitor MBean, 79
 - destination monitor MBean, 72
 - TotalMsgBytesHeldInTransaction attribute,
 - destination monitor MBean, 72
 - TotalMsgBytesInDMQ attribute, destination manager monitor MBean, 79
 - TRANSACTION_COMMIT utility constant, class
 - TransactionNotification, 92
 - TRANSACTION_MANAGER_CONFIG_MBEAN_NAME utility constant
 - class MQObjectName, 26, 88
 - transaction manager configuration MBean, 88-89
 - transaction manager configuration MBean (*Continued*)
 - attribute, 88
 - object name, 88
 - operations, 88-89
 - transaction manager monitor MBean, 89-92
 - attributes, 89-90
 - notification objects, 92
 - notifications, 91-92
 - object name, 89
 - operations, 90-91
 - TRANSACTION_MANAGER_MONITOR_MBEAN_NAME utility constant
 - class MQObjectName, 26, 89
 - TRANSACTION_PREPARE utility constant, class
 - TransactionNotification, 92
 - TRANSACTION_ROLLBACK utility constant, class
 - TransactionNotification, 92
 - TransactionAttributes utility class, 29, 88, 89
 - TransactionID lookup key, transaction, 90
 - TransactionInfo utility class, 29, 90
 - TransactionManager MBean type, 24
 - TransactionNotification class, 29
 - data retrieval method, 92
 - TRANSACTION_COMMIT constant, 92
 - TRANSACTION_PREPARE constant, 92
 - TRANSACTION_ROLLBACK constant, 92
 - utility constants, 91
 - TransactionOperations utility class, 29, 88, 90
 - transactions, 88-92
 - composite data object, lookup keys for (table), 90-91
 - distributed transaction identifier, 90
 - manager configuration MBean, 88-89
 - manager monitor MBean, 89-92
 - state values (table), 91
 - transaction identifier, 89, 90, 92
 - TransactionState utility class, 29, 91
 - COMMITTED constant, 91
 - COMPLETE constant, 91
 - CREATED constant, 91
 - FAILED constant, 91
 - INCOMPLETE constant, 91
 - PREPARED constant, 91
 - ROLLEDBACK constant, 91

TransactionState utility class (*Continued*)

STARTED constant, 91

TIMED_OUT constant, 91

UNKNOWN constant, 91

Transmission Control Protocol (TCP), 24, 25

Transport Layer Security (TLS) protocol, 24, 25

Type attribute

destination configuration MBean, 67, 68, 77

destination monitor MBean, 70, 73

type property (object name), 23

values (table), 23-24

U

UNKNOWN utility constant

class BrokerState, 99

class DestinationState, 73

class LogLevel, 101

class ServiceState, 57

class TransactionState, 91

unquiesce operation, broker configuration MBean, 49

UseDMQ attribute

destination configuration MBean, 67, 68

User attribute, connection monitor MBean, 62

user data object (notifications), 44

User lookup key

message consumer, 86

message producer, 83

transaction, 90

utility classes, 28-29

AdminConnectionConfiguration, 27, 30

BrokerAttributes, 28, 48, 51

BrokerClusterInfo, 29, 95, 98

BrokerOperations, 28, 48

BrokerState, 28, 98

ClusterAttributes, 29, 92, 96

ClusterOperations, 29, 93, 96

ConnectionAttributes, 28, 61, 62, 63, 64

ConnectionOperations, 28, 63, 64, 65

ConsumerAttributes, 29, 84, 85

ConsumerInfo, 29, 86

ConsumerOperations, 29, 84, 85

DestinationAttributes, 28, 66, 70, 75, 77, 78

DestinationLimitBehavior, 29, 68

utility classes (*Continued*)

DestinationOperations, 28, 69, 73, 76, 79

DestinationPauseType, 29, 69, 78

DestinationState, 28, 73

DestinationType, 24, 28, 68, 73, 77, 83, 87

JVMAttributes, 29, 103

LogAttributes, 29, 100

LogLevel, 29, 101

MQObjectName, 20, 25, 28

ProducerAttributes, 29, 81, 82

ProducerInfo, 29, 82

ProducerOperations, 29, 81, 82

ServiceAttributes, 28, 54, 55, 58, 59

ServiceOperations, 28, 54, 57, 58, 60

ServiceState, 28, 56

TransactionAttributes, 29, 88, 89

TransactionInfo, 29, 90

TransactionOperations, 29, 88, 90

TransactionState, 29, 91

utility constants

ALL (class DestinationPauseType), 69, 78

AUTO_ACKNOWLEDGE (interface Session), 87

BROKER_CONFIG_MBEAN_NAME (class
MQObjectName), 26, 48

BROKER_DOWN (class BrokerState), 99

BROKER_MONITOR_MBEAN_NAME (class
MQObjectName), 26, 51BROKER QUIESCE_COMPLETE (class
BrokerNotification), 52BROKER QUIESCE_START (class
BrokerNotification), 52BROKER_SHUTDOWN_START (class
BrokerNotification), 52BROKER TAKEOVER_COMPLETE (class
BrokerNotification), 52, 99BROKER TAKEOVER_FAIL (class
BrokerNotification), 52, 99BROKER TAKEOVER_START (class
BrokerNotification), 52, 99

CLIENT_ACKNOWLEDGE (interface Session), 87

CLUSTER_BROKER_DOWN (class
ClusterNotification), 99CLUSTER_BROKER_JOIN (class
ClusterNotification), 52, 99

utility constants (*Continued*)

CLUSTER_CONFIG_MBEAN_NAME (class MQObjectName), 26, 92
 CLUSTER_MONITOR_MBEAN_NAME (class MQObjectName), 26, 95
 COMMITTED (class TransactionState), 91
 COMPLETE (class TransactionState), 91
 CONNECTION_CLOSE (class ConnectionNotification), 57, 65
 CONNECTION_MANAGER_CONFIG_MBEAN_NAME (class MQObjectName), 26, 63
 CONNECTION_MANAGER_MONITOR_MBEAN_NAME (class MQObjectName), 26, 64
 CONNECTION_OPEN (class ConnectionNotification), 57, 65
 CONNECTION_REJECT (class ConnectionNotification), 57, 65
 CONSUMER_MANAGER_CONFIG_MBEAN_NAME (class MQObjectName), 26, 84
 CONSUMER_MANAGER_MONITOR_MBEAN_NAME (class MQObjectName), 26, 85
 CONSUMERS (class DestinationPauseType), 69, 78
 CONSUMERS_PAUSED (class DestinationState), 73
 CREATED (class TransactionState), 91
 DESTINATION_COMPACT (class DestinationNotification), 74, 80
 DESTINATION_CREATE (class DestinationNotification), 79
 DESTINATION_DESTROY (class DestinationNotification), 79
 DESTINATION_MANAGER_CONFIG_MBEAN_NAME (class MQObjectName), 26, 75
 DESTINATION_MANAGER_MONITOR_MBEAN_NAME (class MQObjectName), 26, 78
 DESTINATION_PAUSE (class DestinationNotification), 74, 79
 DESTINATION_PURGE (class DestinationNotification), 74, 80
 DESTINATION_RESUME (class DestinationNotification), 74, 80
 DUPS_OK_ACKNOWLEDGE (interface Session), 87
 ERROR (class LogLevel), 101
 FAILED (class TransactionState), 91

utility constants (*Continued*)

FLOW_CONTROL (class DestinationLimitBehavior), 68
 INCOMPLETE (class TransactionState), 91
 INFO (class LogLevel), 101
 JVM_MONITOR_MBEAN_NAME (class MQObjectName), 26, 103
 LOG_CONFIG_MBEAN_NAME (class MQObjectName), 26, 100
 LOG_LEVEL_ERROR (class LogNotification), 102
 LOG_LEVEL_INFO (class LogNotification), 102
 LOG_LEVEL_WARNING (class LogNotification), 102
 LOG_MONITOR_MBEAN_NAME (class MQObjectName), 26, 102
 NO_ACKNOWLEDGE (interface Session), 87
 NONE (class LogLevel), 101
 OPERATING (class BrokerState), 98
 PAUSED (class DestinationState), 73
 PAUSED (class ServiceState), 57
 PREPARED (class TransactionState), 91
 PRODUCER_MANAGER_CONFIG_MBEAN_NAME (class MQObjectName), 26, 81
 PRODUCER_MANAGER_MONITOR_MBEAN_NAME (class MQObjectName), 26, 81
 PRODUCERS (class DestinationPauseType), 69, 78
 PRODUCERS_PAUSED (class DestinationState), 73
 QUEUE (class DestinationType), 24, 68, 73, 78, 83, 87
 QUIESCE_COMPLETE (class BrokerState), 99
 QUIESCE_STARTED (class BrokerState), 99
 QUIESCED (class ServiceState), 57
 REJECT_NEWEST (class DestinationLimitBehavior), 69
 REMOVE_LOW_PRIORITY (class DestinationLimitBehavior), 68
 REMOVE_OLDEST (class DestinationLimitBehavior), 68
 ROLLEDBACK (class TransactionState), 91
 RUNNING (class DestinationState), 73
 RUNNING (class ServiceState), 57
 SERVICE_MANAGER_CONFIG_MBEAN_NAME (class MQObjectName), 26, 58
 SERVICE_MANAGER_MONITOR_MBEAN_NAME (class MQObjectName), 26, 59

utility constants (*Continued*)

SERVICE_PAUSE (class ServiceNotification), 57, 60

SERVICE_RESUME (class ServiceNotification), 57, 60

SESSION_TRANSACTED (interface Session), 87

SHUTDOWN_STARTED (class BrokerState), 99

STARTED (class TransactionState), 91

TAKEOVER_COMPLETE (class BrokerState), 99

TAKEOVER_FAILED (class BrokerState), 99

TAKEOVER_STARTED (class BrokerState), 99

TIMED_OUT (class TransactionState), 91

TOPIC (class DestinationType), 24, 68, 73, 78, 83, 87

TRANSACTION_COMMIT (class TransactionNotification), 92

TRANSACTION_MANAGER_CONFIG_MBEAN_NAME (class MQObjectName), 26, 88

TRANSACTION_MANAGER_MONITOR_MBEAN_NAME (class MQObjectName), 26, 89

TRANSACTION_PREPARE (class TransactionNotification), 92

TRANSACTION_ROLLBACK (class TransactionNotification), 92

UNKNOWN (class BrokerState), 99

UNKNOWN (class DestinationState), 73

UNKNOWN (class LogLevel), 101

UNKNOWN (class ServiceState), 57

UNKNOWN (class TransactionState), 91

WARNING (class LogLevel), 101

utility methods

createConnectionConfig (class MQObjectName), 26, 61

createConnectionMonitor (class MQObjectName), 26, 62

createDestinationConfig (class MQObjectName), 26, 66

createDestinationMonitor (class MQObjectName), 26, 70

createServiceConfig (class MQObjectName), 26, 53

createServiceMonitor (class MQObjectName), 26, 55

V

Version attribute

broker configuration MBean, 48

broker monitor MBean, 51

W

WARNING utility constant, class LogLevel, 101

X

XID, *See* distributed transaction identifier

XID lookup key, transaction, 90

