



Open Message Queue 4.5 Developer's Guide for JMX Clients

Oracle Corporation
500 Oracle Parkway
Redwood City, CA 94065
U.S.A.

Part No: 821-2481-12
July 2011

Copyright © 2010, 2011, Oracle and/or its affiliates. All rights reserved.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS

Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, duplication, disclosure, modification, and adaptation shall be subject to the restrictions and license terms set forth in the applicable Government contract, and, to the extent applicable by the terms of the Government contract, the additional rights set forth in FAR 52.227-19, Commercial Computer Software License (December 2007). Oracle America, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

Contents

Preface	11
1 Introduction to JMX Programming for Message Queue Clients	19
JMX Architecture	19
Message Queue MBeans	20
Resource MBeans	21
Manager MBeans	22
Object Names	23
2 Using the JMX API	29
Interface Packages	29
Utility Classes	30
Connecting to the MBean Server	31
Obtaining a JMX Connector from an Admin Connection Factory	32
Obtaining a JMX Connector Without Using an Admin Connection Factory	33
Using MBeans	34
Accessing MBean Attributes	34
Invoking MBean Operations	39
Receiving MBean Notifications	47
3 Message Queue MBean Reference	51
Brokers	51
Broker Configuration	51
Broker Monitor	55
Connection Services	58
Service Configuration	58
Service Monitor	60

Service Manager Configuration	63
Service Manager Monitor	64
Connections	66
Connection Configuration	66
Connection Monitor	67
Connection Manager Configuration	68
Connection Manager Monitor	69
Destinations	71
Destination Configuration	71
Destination Monitor	75
Destination Manager Configuration	81
Destination Manager Monitor	85
Message Producers	87
Producer Manager Configuration	87
Producer Manager Monitor	88
Message Consumers	90
Consumer Manager Configuration	90
Consumer Manager Monitor	91
Transactions	95
Transaction Manager Configuration	95
Transaction Manager Monitor	96
Broker Clusters	99
Cluster Configuration	99
Cluster Monitor	103
Logging	108
Log Configuration	108
Log Monitor	110
Java Virtual Machine	111
JVM Monitor	111
A Alphabetical Reference	113
Index	125

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	25
TABLE 1-6	Example Object Names	25
TABLE 1-7	Utility Constants and Methods for Object Names	26
TABLE 2-1	Message Queue JMX Utility Classes	30
TABLE 3-1	Broker Configuration Attributes	52
TABLE 3-2	Broker Configuration Operations	53
TABLE 3-3	Broker Configuration Notification	55
TABLE 3-4	Broker Monitor Attributes	56
TABLE 3-5	Broker Monitor Notifications	57
TABLE 3-6	Data Retrieval Methods for Broker Monitor Notifications	57
TABLE 3-7	Connection Service Names for Service Configuration MBeans	59
TABLE 3-8	Service Configuration Attributes	59
TABLE 3-9	Service Configuration Operations	60
TABLE 3-10	Service Configuration Notification	60
TABLE 3-11	Connection Service Names for Service Monitor MBeans	60
TABLE 3-12	Service Monitor Attributes	61
TABLE 3-13	Connection Service State Values	62
TABLE 3-14	Service Monitor Operations	62
TABLE 3-15	Service Monitor Notifications	63
TABLE 3-16	Data Retrieval Method for Service Monitor Notifications	63
TABLE 3-17	Service Manager Configuration Attributes	63
TABLE 3-18	Service Manager Configuration Operations	64
TABLE 3-19	Service Manager Monitor Attributes	64
TABLE 3-20	Service Manager Monitor Operation	65

TABLE 3-21	Service Manager Monitor Notifications	65
TABLE 3-22	Data Retrieval Method for Service Manager Monitor Notifications	66
TABLE 3-23	Connection Configuration Attribute	67
TABLE 3-24	Connection Monitor Attributes	67
TABLE 3-25	Connection Monitor Operations	68
TABLE 3-26	Connection Manager Configuration Attribute	68
TABLE 3-27	Connection Manager Configuration Operations	69
TABLE 3-28	Connection Manager Monitor Attributes	69
TABLE 3-29	Connection Manager Monitor Operation	70
TABLE 3-30	Connection Manager Monitor Notifications	70
TABLE 3-31	Data Retrieval Methods for Connection Manager Monitor Notifications	70
TABLE 3-32	Destination Configuration Attributes	71
TABLE 3-33	Destination Configuration Type Values	74
TABLE 3-34	Destination Limit Behaviors	74
TABLE 3-35	Destination Configuration Operations	74
TABLE 3-36	Destination Pause Types	75
TABLE 3-37	Destination Configuration Notification	75
TABLE 3-38	Destination Monitor Attributes	76
TABLE 3-39	Destination Monitor Type Values	79
TABLE 3-40	Destination State Values	79
TABLE 3-41	Destination Monitor Operations	80
TABLE 3-42	Destination Monitor Notifications	81
TABLE 3-43	Data Retrieval Methods for Destination Monitor Notifications	81
TABLE 3-44	Destination Manager Configuration Attributes	82
TABLE 3-45	Destination Manager Configuration Operations	83
TABLE 3-46	Destination Manager Configuration Type Values	84
TABLE 3-47	Destination Manager Pause Types	84
TABLE 3-48	Destination Manager Configuration Notification	85
TABLE 3-49	Destination Manager Monitor Attributes	85
TABLE 3-50	Destination Manager Monitor Operation	86
TABLE 3-51	Destination Manager Monitor Notifications	86
TABLE 3-52	Data Retrieval Methods for Destination Manager Monitor Notifications	86
TABLE 3-53	Producer Manager Configuration Attribute	87
TABLE 3-54	Producer Manager Configuration Operation	88
TABLE 3-55	Producer Manager Monitor Attribute	88
TABLE 3-56	Producer Manager Monitor Operations	89

TABLE 3-57	Lookup Keys for Message Producer Information	89
TABLE 3-58	Message Producer Destination Types	90
TABLE 3-59	Consumer Manager Configuration Attribute	91
TABLE 3-60	Consumer Manager Configuration Operations	91
TABLE 3-61	Consumer Manager Monitor Attribute	92
TABLE 3-62	Consumer Manager Monitor Operations	92
TABLE 3-63	Lookup Keys for Message Consumer Information	93
TABLE 3-64	Message Consumer Destination Types	94
TABLE 3-65	Acknowledgment Modes	95
TABLE 3-66	Transaction Manager Configuration Attribute	96
TABLE 3-67	Transaction Manager Configuration Operations	96
TABLE 3-68	Transaction Manager Monitor Attributes	97
TABLE 3-69	Transaction Manager Monitor Operations	97
TABLE 3-70	Lookup Keys for Transaction Information	98
TABLE 3-71	Transaction State Values	98
TABLE 3-72	Transaction Manager Monitor Notifications	99
TABLE 3-73	Data Retrieval Method for Transaction Manager Monitor Notifications	99
TABLE 3-74	Cluster Configuration Attributes	100
TABLE 3-75	Cluster Configuration Operations	101
TABLE 3-76	Lookup Keys for Cluster Configuration Information	103
TABLE 3-77	Lookup Keys for changeMasterBroker	103
TABLE 3-78	Cluster Configuration Notification	103
TABLE 3-79	Cluster Monitor Attributes	104
TABLE 3-80	Cluster Monitor Operations	105
TABLE 3-81	Lookup Keys for Cluster Monitor Information	106
TABLE 3-82	Broker State Values	107
TABLE 3-83	Cluster Monitor Notifications	107
TABLE 3-84	Data Retrieval Methods for Cluster Monitor Notifications	108
TABLE 3-85	Log Configuration Attributes	109
TABLE 3-86	Log Configuration Logging Levels	109
TABLE 3-87	Log Configuration Notification	110
TABLE 3-88	Log Monitor Notifications	110
TABLE 3-89	Data Retrieval Methods for Log Monitor Notifications	111
TABLE 3-90	JVM Monitor Attributes	111
TABLE A-1	Alphabetical List of MBean Attributes	113
TABLE A-2	Alphabetical List of MBean Operations	119

TABLE A-3 Alphabetical List of MBean Notifications 122

Examples

EXAMPLE 2-1	Obtaining a JMX Connector from an Admin Connection Factory	32
EXAMPLE 2-2	Configuring an Admin Connection Factory	33
EXAMPLE 2-3	Obtaining a JMX Connector Without Using an Admin Connection Factory ...	33
EXAMPLE 2-4	Getting an Attribute Value	35
EXAMPLE 2-5	Getting Multiple Attribute Values	36
EXAMPLE 2-6	Setting an Attribute Value	37
EXAMPLE 2-7	Setting Multiple Attribute Values	38
EXAMPLE 2-8	Invoking an Operation	40
EXAMPLE 2-9	Invoking an Operation with Parameters	41
EXAMPLE 2-10	Combining Operations and Attributes	43
EXAMPLE 2-11	Using a Composite Data Object	45
EXAMPLE 2-12	Notification Listener	48
EXAMPLE 2-13	Registering a Notification Listener	49

Preface

This *Developer's Guide for JMX Clients* describes the application programming interface provided in Open Message Queue for programmatically configuring and monitoring Message Queue resources in conformance with the Java Management Extensions (JMX). 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*; the API described here makes the same administrative functionality available programmatically from within a running client application. Broker properties and command-line options that support the JMX API are described in the *Message Queue Administration Guide*.

This preface consists of the following sections:

- “Who Should Use This Book” on page 11
- “Before You Read This Book” on page 11
- “How This Book Is Organized” on page 12
- “Documentation Conventions” on page 12
- “Related Documentation” on page 15
- “Documentation, Support, and Training” on page 17
- “Third-Party Web Site References” on page 17

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

Table P–1 describes the contents of this manual.

TABLE P–1 Book Contents

Chapter	Description
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.

Documentation Conventions

This section describes the following conventions used in Message Queue documentation:

- “Typographic Conventions” on page 12
- “Symbol Conventions” on page 13
- “Shell Prompt Conventions” on page 14
- “Directory Variable Conventions” on page 14

Typographic Conventions

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

TABLE P-2 Typographic Conventions

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

Symbol Conventions

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

TABLE P-3 Symbol Conventions

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

Shell Prompt Conventions

The following table shows the conventions used in Message Queue documentation for the default UNIX system prompt and superuser prompt for the C shell, Bourne shell, Korn shell, and for the Windows operating system.

TABLE P-4 Shell Prompt Conventions

Shell	Prompt
C shell on UNIX, Linux, or AIX	<i>machine-name%</i>
C shell superuser on UNIX, Linux, or AIX	<i>machine-name#</i>
Bourne shell and Korn shell on UNIX, Linux, or AIX	\$
Bourne shell and Korn shell superuser on UNIX, Linux, or AIX	#
Windows command line	C:\>

Directory Variable Conventions

Message Queue documentation makes use of three directory variables; two of which represent environment variables needed by Message Queue. (How you set the environment variables varies from platform to platform.)

The following table describes the directory variables that might be found in this book and how they are used. Some of these variables refer to the directory *mqInstallHome*, which is the directory where Message Queue is installed to when using the installer or unzipped to when using a zip-based distribution.

Note – In this book, directory variables are shown without platform-specific environment variable notation or syntax (such as \$IMQ_HOME on UNIX). Non-platform-specific path names use UNIX directory separator (/) notation.

TABLE P-5 Directory Variable Conventions

Variable	Description
IMQ_HOME	The Message Queue home directory: <ul style="list-style-type: none">For installations of Message Queue bundled with GlassFish Server, IMQ_HOME is <i>as-install-parent/mq</i>, where <i>as-install-parent</i> is the parent directory of the GlassFish Server base installation directory, <i>glassfish3</i> by default.For installations of Open Message Queue, IMQ_HOME is <i>mqInstallHome/mq</i>.

TABLE P-5 Directory Variable Conventions (Continued)

Variable	Description
IMQ_VARHOME	<p>The directory in which Message Queue temporary or dynamically created configuration and data files are stored; IMQ_VARHOME can be explicitly set as an environment variable to point to any directory or will default as described below:</p> <ul style="list-style-type: none"> For installations of Message Queue bundled with GlassFish Server, IMQ_VARHOME defaults to <i>as-install-parent/glassfish/domains/domain1/imq</i>. For installations of Open Message Queue, IMQ_HOME defaults to <i>mqInstallHome/var/mq</i>.
IMQ_JAVAHOME	<p>An environment variable that points to the location of the Java runtime environment (JRE) required by Message Queue executable files. By default, Message Queue looks for and uses the latest JDK, but you can optionally set the value of IMQ_JAVAHOME to wherever the preferred JRE resides.</p>

Related Documentation

The information resources listed in this section provide further information about Message Queue in addition to that contained in this manual. The section covers the following resources:

- “Message Queue Documentation Set” on page 15
- “Java Message Service (JMS) Specification” on page 16
- “JavaDoc” on page 16
- “Example Client Applications” on page 16
- “Online Help” on page 17

Message Queue Documentation Set

The documents that constitute the Message Queue documentation set are listed in the following table in the order in which you might normally use them. These documents are available through the Open Server documentation web site at

<http://www.oracle.com/technetwork/indexes/documentation/index.html>

TABLE P-6 Message Queue Documentation Set

Document	Audience	Description
<i>Open Message Queue 4.5 Technical Overview</i>	Developers and administrators	Describes Message Queue concepts, features, and components.
<i>Open Message Queue 4.5 Release Notes</i>	Developers and administrators	Includes descriptions of new features, limitations, and known bugs, as well as technical notes.

TABLE P-6 Message Queue Documentation Set (Continued)

Document	Audience	Description
<i>Open Message Queue 4.5 Administration Guide</i>	Administrators, also recommended for developers	Provides background and information needed to perform administration tasks using Message Queue administration tools.
<i>Open Message Queue 4.5 Developer's Guide for Java Clients</i>	Developers	Provides a quick-start tutorial and programming information for developers of Java client programs using the Message Queue implementation of the JMS or SOAP/JAXM APIs.
<i>Open Message Queue 4.5 Developer's Guide for C Clients</i>	Developers	Provides programming and reference documentation for developers of C client programs using the Message Queue C implementation of the JMS API (C-API).
<i>Open Message Queue 4.5 Developer's Guide for JMX Clients</i>	Administrators	Provides programming and reference documentation for developers of JMX client programs using the Message Queue JMX API.

Java Message Service (JMS) Specification

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

<http://www.oracle.com/technetwork/java/jms/index.html>

JavaDoc

JMS and Message Queue API documentation in JavaDoc format is included in Message Queue installations at `IMQ_HOME/javadoc/index.html`. This documentation can be viewed in any HTML browser. It includes standard JMS API documentation as well as Message Queue-specific APIs.

Example Client Applications

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

Example Java Client Applications

Example Java client applications are included in Message Queue installations at `IMQ_HOME/examples`. See the README files located in this directory and its subdirectories for descriptive information about the example applications.

Example C Client Programs

Example C client applications are included in Message Queue installations at `IMQ_HOME/examples/C`. See the README files located in this directory and its subdirectories for descriptive information about the example applications.

Example JMX Client Programs

Example Java Management Extensions (JMX) client applications are included in Message Queue installations at `IMQ_HOME/examples/jmx`. See the README files located in this directory and its subdirectories for descriptive information about the example applications.

Online Help

Online help is available for the Message Queue command line utilities; for details, see Chapter 16, “Command Line Reference,” in *Open Message Queue 4.5 Administration Guide* for details. The Message Queue graphical user interface (GUI) administration tool, the Administration Console, also includes a context-sensitive help facility; see the section “Administration Console Online Help” in Chapter 2, “Quick-Start Tutorial,” in *Open Message Queue 4.5 Administration Guide*.

Documentation, Support, and Training

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

- Documentation (<http://www.oracle.com/technetwork/indexes/documentation/index.html>)
- Support (<http://www.oracle.com/us/support/044752.html>)
- Training (http://education.oracle.com/pls/web_prod-plq-dad/db_pages.getpage?page_id=315)

Third-Party Web Site References

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

Note – Oracle is not responsible for the availability of third-party Web sites mentioned in this manual. Oracle does not endorse and is not responsible or liable for any content, advertising, products, or other materials available on or through such sites or resources. Oracle 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 the use of or reliance on any such content, goods, or services available on or through such sites or resources.

Introduction to JMX Programming for Message Queue Clients

While Open Message Queue's Administration Console and command line administration utilities allow an administrator to interactively configure and monitor Message Queue resources (such as 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 JMX API to programmatically perform the configuration and monitoring operations 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 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 brokers, connections, 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 23](#) 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 connection 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.
desttype	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: <pre>myTopic temporary_destination://queue/129.145.180.99/63008/1</pre>
id	Resource identifier Applies only to MBeans of the following types: <ul style="list-style-type: none"> Connection configuration Connection monitor 	Example: <pre>7853717387765338368</pre>

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

TABLE 1-2 Message Queue MBean Types *(Continued)*

Value	Description
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
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
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	<code>com.sun.messaging.jms.server:type=Broker,subtype=Config</code>
Service manager monitor	<code>com.sun.messaging.jms.server:type=ServiceManager,subtype=Monitor</code>
Connection configuration	<code>com.sun.messaging.jms.server:type=Connection,subtype=Config,id=7853717387765338368</code>
Destination monitor	<code>com.sun.messaging.jms.server:type=Destination,subtype=Monitor,desttype=t,name="MyQueue"</code>

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>

TABLE 1-7 Utility Constants and Methods for Object Names (Continued)

MBean Type	Utility Constant or Method
Log configuration	MQObjectName.LOG_CONFIG_MBEAN_NAME
Log monitor	MQObjectName.LOG_MONITOR_MBEAN_NAME
JVM monitor	MQObjectName.JVM_MONITOR_MBEAN_NAME

Using the JMX API

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

- [“Interface Packages” on page 29](#)
- [“Utility Classes” on page 30](#)
- [“Connecting to the MBean Server” on page 31](#)
- [“Using MBeans” on page 34](#)

Interface Packages

The Message Queue 4.5 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 31](#)), 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 30](#)) 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 in the `IMQ_HOME/lib` directory.

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 2–1](#) 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 2–1 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
<code>DestinationLimitBehavior</code>	Names of destination limit behaviors
<code>DestinationPauseType</code>	Constants related to destination pause type
<code>ProducerAttributes</code>	Names of message producer attributes
<code>ProducerOperations</code>	Names of message producer operations

TABLE 2-1 Message Queue JMX Utility Classes (Continued)

Class	Description
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 connection*, accessed by means of a *JMX connector*. Message Queue brokers use the standard JMX infrastructure provided with the Java Development Kit (JDK) 1.5, which uses remote method invocation (RMI) for communicating between client and server. Once you obtain 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. This infrastructure is describe in “JMX Connection Infrastructure” in *Open Message Queue 4.5 Administration Guide*.

For convenience, Message Queue provides an *admin 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 Admin Connection Factory

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

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

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

// Create admin 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 jmxcr = acf.createConnection();

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

[Example 2-2](#) shows how to reconfigure an admin 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 `AdminConnectionConfiguration`, 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 Admin Connection Factory

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

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

// Configure for specific broker address
acf.setProperty(AdminConnectionFactoryConfiguration.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 Admin 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 admin connection factory in order to avoid dependency on Message Queue-specific classes.

EXAMPLE 2-3 Obtaining a JMX Connector Without Using an Admin 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);

// Get 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);
```

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

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

The `JMXConnectorFactory.connect` method accepts two parameters:

- A *JMX service URL*.

The JMX service URL is an address used for obtaining the JMX connector. It can either specify the location of a JMX connector stub in an RMI registry or contain a connector stub as a serialized object. These options, and the format of the address, are described in “The JMX Service URL” in *Open Message Queue 4.5 Administration Guide*

- An optional *environment* parameter.

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.

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 71).

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 admin 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
            ObjectName destConfigName
                = 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
            jmxcr.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 71](#)).

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 admin 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 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
            jmx.close();
        }
    }
}
```

EXAMPLE 2-5 Getting Multiple Attribute Values (Continued)

```

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

public class SetAttrValue
{
    public static void main (String[] args)
    {
        try
        { // Create admin 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);
        }
    }
}

```

EXAMPLE 2-6 Setting an Attribute Value *(Continued)*

```
        // Close JMX connector
        jmxcc.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 admin connection factory
          AdminConnectionFactory acf = new AdminConnectionFactory();

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

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

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

EXAMPLE 2-7 Setting Multiple Attribute Values (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);

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

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

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

```

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 58).

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 admin 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 serviceName = MBeanServerConnection.createServiceConfig("jms");

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

            // Close JMX connector
            jmx.close();
        }
        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 81](#)) 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 admin 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 destMgrConfigName
                = new ObjectName(MQObjectName.DESTINATION_MANAGER_CONFIG_MBEAN_NAME);

            // 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);
```

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

```
// 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
jmxc.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 85](#)) 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 admin 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 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);
```

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

```
// 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 );

    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 91 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 admin 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 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;
        }
    }
}
```

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

```
// 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();
}

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)
    {
        try
        { // Create admin 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 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();
        }
    }
}
```

EXAMPLE 2-13 Registering a Notification Listener *(Continued)*

```
        finally
        { if ( jmxrc != null )
          { try
            { jmxrc.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:

- “Brokers” on page 51
- “Connection Services” on page 58
- “Connections” on page 66
- “Destinations” on page 71
- “Message Producers” on page 87
- “Message Consumers” on page 90
- “Transactions” on page 95
- “Broker Clusters” on page 99
- “Logging” on page 108
- “Java Virtual Machine” on page 111

Brokers

This section describes the MBeans used for managing brokers:

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

The following subsections describe each of these MBeans in detail.

Broker Configuration

The *broker configuration MBean* is used for configuring a 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	Broker identifier Must be a unique alphanumeric string of no more than $n - 13$ characters, where n is the maximum table name length allowed by the database. No two running brokers may have the same broker identifier. For brokers using a JDBC-based persistent data store, this string is appended to the names of all database tables to make them unique in the case where more than one broker instance is using the same database. If a database is not used as the persistent data store, the value of this attribute is <code>null</code> . Note – For high-availability brokers, database table names use the <code>ClusterID</code> attribute (see Table 3–74) instead.
Version	String	No	Broker version
InstanceName	String	No	Broker instance name Example: <code>imqbroker</code>
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>).
getProperty	<i>propertyName</i> (String)	String	Get value of configuration property The desired property is designated by its name (<i>propertyName</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:

¹ HA clusters only

TABLE 3-2 Broker Configuration Operations (Continued)

Name	Parameters	Result Type	Description
			Service monitor NumConnectionsOpened NumConnectionsRejected NumMsgsIn NumMsgsOut MsgBytesIn MsgBytesOut NumPktsIn NumPktsOut PktBytesIn PktBytesOut
			Service manager monitor NumMsgsIn NumMsgsOut MsgBytesIn MsgBytesOut NumPktsIn NumPktsOut PktBytesIn PktBytesOut
			Connection manager monitor NumConnectionsOpened NumConnectionsRejected
			Destination monitor PeakNumConsumers AvgNumConsumers PeakNumActiveConsumers AvgNumActiveConsumers PeakNumBackupConsumers AvgNumBackupConsumers PeakNumMsgs AvgNumMsgs NumMsgsIn NumMsgsOut MsgBytesIn MsgBytesOut PeakMsgBytes PeakTotalMsgBytes AvgTotalMsgBytes

TABLE 3-2 Broker Configuration Operations (Continued)

Name	Parameters	Result Type	Description
			Transaction manager monitor NumTransactionsCommitted NumTransactionsRollback

Notification

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

TABLE 3-3 Broker Configuration Notification

Name	Description
jmx.attribute.change	Attribute value changed

Broker Monitor

The *broker monitor MBean* is used for monitoring a 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
BrokerID	String	No	<p>Broker identifier</p> <p>Must be a unique alphanumeric string of no more than $n - 13$ characters, where n is the maximum table name length allowed by the database. 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 in the case where more than one broker instance is using the same 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 3-79) instead.</p>
Version	String	No	Broker version
InstanceName	String	No	Broker instance name
Port	Integer	No	Port number of Port Mapper
ResourceState	String	No	<p>Current broker resource state:</p> <ul style="list-style-type: none"> green: < 80% memory utilization yellow: 80–90% memory utilization orange: 90–98% memory utilization red: > 98% memory utilization <p>Note – The threshold values shown are the default thresholds for triggering the various states; these can be changed by setting the broker configuration properties</p> <ul style="list-style-type: none"> <code>imq.green.threshold</code> <code>imq.yellow.threshold</code> <code>imq.orange.threshold</code> <code>imq.red.threshold</code>
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.broker.resource.state.change</code>	<code>BrokerNotification.BROKER_RESOURCE_STATE_CHANGE</code>	Broker's resource state has changed
<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 3-84 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 <i>hostName:portNumber</i> Example: <code>host1:3000</code>
<code>getFailedBrokerID</code> ¹	String	Broker identifier of broker being taken over

¹ HA clusters only

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

Method	Result Type	Description
getOldResourceState	String	Broker's previous resource state: green: < 80% memory utilization yellow: 80–90% memory utilization orange: 90–98% memory utilization red: > 98% memory utilization Note – The threshold values shown are the default thresholds for triggering the various states; these can be changed by setting the broker configuration properties imq.green.threshold imq.yellow.threshold imq.orange.threshold imq.red.threshold
getNewResourceState	String	Broker's new resource state (see getOldResourceState, above, for possible values)
getHeapMemoryUsage	MemoryUsage	Broker's current heap memory usage The value returned is an object of class MemoryUsage (defined in the package java.lang.management).

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
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 (jms, ssljms, admin, and ssladmin 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 (jms and admin 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 (jms, ssljms, httpjms, and httpsjms services only)
resume	None	None	Resume service (jms, ssljms, httpjms, and httpsjms services only)

Notification

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

TABLE 3–10 Service Configuration Notification

Name	Description
jmx.attribute.change	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
jms	Normal	TCP
ssljms	Normal	TLS (SSL-based security)
httpjms	Normal	HTTP
httpsjms	Normal	HTTPS (SSL-based security)

TABLE 3–11 Connection Service Names for Service Monitor MBeans *(Continued)*

Service Name	Service Type	Protocol Type
admin	Admin	TCP
ssladmin	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

TABLE 3–12 Service Monitor Attributes (Continued)

Name	Type	Settable?	Description
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	<code>ServiceState.RUNNING</code>	<code>RUNNING</code>	Service running
1	<code>ServiceState.PAUSED</code>	<code>PAUSED</code>	Service paused
2	<code>ServiceState.QUIESCED</code>	<code>QUIESCED</code>	Service quiesced
–1	<code>ServiceState.UNKNOWN</code>	<code>UNKNOWN</code>	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
------	------	-----------	-------------

TABLE 3-17 Service Manager Configuration Attributes (Continued)

MinThreads	Integer	No	Total minimum number of threads for all active services
MaxThreads	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
<code>getServices</code>	None	<code>ObjectName[]</code>	Object names of service configuration MBeans for all services
<code>pause</code>	None	None	Pause all services except <code>admin</code> and <code>ssladmin</code>
<code>resume</code>	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
<code>NumServices</code>	Integer	No	Number of connection services
<code>NumActiveThreads</code>	Integer	No	Total current number of threads actively handling connections for all services

TABLE 3–19 Service Manager Monitor Attributes (Continued)

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

TABLE 3–24 Connection Monitor Attributes (Continued)

Name	Type	Settable?	Description
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
getService	None	ObjectName	Object name of service monitor MBean for associated connection service
getTemporaryDestinations	None	ObjectName[]	Object names of destination monitor MBeans for all associated temporary destinations
getProducerIDs	None	String[]	Producer identifiers of all associated message producers
getConsumerIDs	None	String[]	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
<code>NumConnections</code>	Integer	No	Current number of connections
<code>NumConnectionsOpened</code>	Long	No	Cumulative number of connections opened since broker started
<code>NumConnectionsRejected</code>	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
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.

TABLE 3–32 Destination Configuration Attributes (Continued)

Name	Type	Settable?	Description
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.
ValidateXMLSchemaEnabled	Boolean	Yes	XML schema validation is enabled? If set to <code>false</code> or not set, then XML schema validation is not enabled for the destination.
XMLSchemaURIList	String	Yes	Space separated list of XML schema document (XSD) URI strings The URIs point to the location of one or more XSDs to use for XML schema validation, if enabled. Use double quotes around this value if multiple URIs are specified. Example: “ <code>http://foo/flap.xsd http://test.com/test.xsd</code> ” If this property is not set or null and XML validation is enabled, XML validation is performed using a DTD specified in the XML document.
ReloadXMLSchemaOnFailure	Boolean	Yes	Reload XML schema on failure enabled? If set to <code>false</code> or not set, then the schema is not reloaded if validation fails.

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
<code>FLOW_CONTROL</code>	<code>DestinationLimitBehavior.FLOW_CONTROL</code>	Slow down producers
<code>REMOVE_OLDEST</code>	<code>DestinationLimitBehavior.REMOVE_OLDEST</code>	Throw out oldest messages
<code>REMOVE_LOW_PRIORITY</code>	<code>DestinationLimitBehavior.REMOVE_LOW_PRIORITY</code>	Throw out lowest-priority messages according to age; no notice to producing client
<code>REJECT_NEWEST</code>	<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>	<i>pauseType</i> (String)	None	Pause message delivery See Table 3–36 for possible values of <i>pauseType</i> .
<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

TABLE 3–35 Destination Configuration Operations (Continued)

Name	Parameters	Result Type	Description
compact 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?
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 (<code>jconsole</code>). 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 <code>NumActiveConsumers</code> .
NumWildcardProducers	Integer	No	Current number of wildcard message producers associated with the destination For topic destinations only.
NumWildcardConsumers	Integer	No	Current number of wildcard message consumers associated with the destination For topic destinations only.

¹ Temporary destinations only

TABLE 3-38 Destination Monitor Attributes (Continued)

Name	Type	Settable?	Description
NumWildcards	Integer	No	Current number of wildcard message producers and wildcard message consumers associated with the destination For topic destinations only.
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 <code>PeakNumActiveConsumers</code> .
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 <code>AvgNumActiveConsumers</code> .
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 <code>NumConsumers</code> .
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 <code>PeakNumConsumers</code> .
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 <code>AvgNumConsumers</code> .
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

² Queue destinations only

TABLE 3-38 Destination Monitor Attributes *(Continued)*

Name	Type	Settable?	Description
NumMsgs	Long	No	Current number of messages stored in memory and persistent store Does not include messages held in transactions.
NumMsgsRemote	Long	No	Current number of messages stored in memory and persistent store that were produced to a remote broker in a cluster. This number does not include messages included 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
NextMessageID	String	No	JMS Message ID of the next message to be delivered to any consumer
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.
TotalMsgBytesRemote	Long	No	Current total size in bytes of messages stored in memory and persistent store that were produced to a remote broker in a cluster. This value does not include messages included in transactions.
TotalMsgBytesHeldInTransaction	Long	No	Current total size in bytes of messages being held in memory and persistent store in uncommitted transactions
PeakTotalMsgBytes	Long	No	Peak total size in bytes of messages stored in memory and persistent store since broker started

TABLE 3–38 Destination Monitor Attributes *(Continued)*

Name	Type	Settable?	Description
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
getConsumerWildcards	none	String[]	Wildcard strings used by current consumers associated with the destination For topic destinations only.
getProducerWildcards	none	String[]	Wildcard strings used by current producers associated with the destination For topic destinations only.
getWildcards	none	String[]	Wildcard strings used by current consumers and producers associated with the destination For topic destinations only.
getNumWildcardConsumers	wildcard-String	Integer	Number of current consumers associated with the destination that are using the specified wildcard string For topic destinations only.

¹ Temporary destinations only² Queue destinations only

TABLE 3–41 Destination Monitor Operations (Continued)

Name	Parameters	Result Type	Description
getNumWildcardProducers	wildcard-String	Integer	Number of current producers associated with the destination that are using the specified wildcard string For topic 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
AutoCreateQueues	Boolean	Yes	Allow auto-creation of queue destinations?
AutoCreateTopics	Boolean	Yes	Allow auto-creation of topic destinations?
NumDestinations	Integer	No	Current total number of destinations
MaxNumMsgs	Long	Yes	Maximum total 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 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 true, only the message header and property data will be saved.

¹ Auto-created queue destinations only

TABLE 3–44 Destination Manager Configuration Attributes (Continued)

Name	Type	Settable?	Description
LogDeadMsgs	Boolean	Yes	<p>Log information about dead messages?</p> <p>If <code>true</code>, the following events will be logged:</p> <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.

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
<code>getDestinations</code>	None	<code>ObjectName[]</code>	Object names of destination configuration MBeans for all current destinations
<code>create</code>	<i>destinationType</i> (String) <i>destinationName</i> (String) <i>destinationAttributes</i> (AttributeList)	None	<p>Create destination with specified type, name, and attributes</p> <p>The <i>destinationType</i> and <i>destinationName</i> parameters are required, but <i>destinationAttributes</i> may be <code>null</code>.</p> <p>See Table 3–46 for possible values of <i>destinationType</i>.</p> <p>The <i>destinationAttributes</i> list may include any of the attributes listed in Table 3–32 except <code>Name</code> and <code>Type</code>. The names of these attributes are defined as static constants in the utility class <code>DestinationAttributes</code>.</p>
<code>create</code>	<i>destinationType</i> (String) <i>destinationName</i> (String)	None	<p>Create destination with specified type and name</p> <p>Equivalent to <code>create(destinationType, destinationName, null)</code>.</p> <p>See Table 3–46 for possible values of <i>destinationType</i>.</p>

TABLE 3–45 Destination Manager Configuration Operations (Continued)

Name	Parameters	Result Type	Description
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
<code>NumDestinations</code>	Integer	No	Current total number of destinations
<code>NumMsgs</code>	Long	No	Current total number of messages stored in memory and persistent store for all destinations Does not include messages held in transactions.
<code>TotalMsgBytes</code>	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.
<code>NumMsgsInDMQ</code>	Long	No	Current number of messages stored in memory and persistent store for dead message queue
<code>TotalMsgBytesInDMQ</code>	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
<code>getDestinations</code>	None	<code>ObjectName[]</code>	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
<code>mq.destination.create</code>	<code>DestinationNotification.DESTINATION_CREATE</code>	Destination created
<code>mq.destination.destroy</code>	<code>DestinationNotification.DESTINATION_DESTROY</code>	Destination destroyed
<code>mq.destination.pause</code>	<code>DestinationNotification.DESTINATION_PAUSE</code>	Destination paused
<code>mq.destination.resume</code>	<code>DestinationNotification.DESTINATION_RESUME</code>	Destination resumed
<code>mq.destination.compact</code>	<code>DestinationNotification.DESTINATION_COMPACT</code>	Destination compacted
<code>mq.destination.purge</code>	<code>DestinationNotification.DESTINATION_PURGE</code>	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?

TABLE 3–52 Data Retrieval Methods for Destination Manager Monitor Notifications (Continued)

Method	Result Type	Description
getPauseType	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
<code>getProducerIDs</code>	None	<code>String[]</code>	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
<code>NumWildcardProducers</code>	Integer	No	Number of wildcard message producers associated with the broker

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
getProducerIDs	None	String[]	Producer identifiers of all current message producers
getProducerInfoByID	<i>producerID</i> (String)	CompositeData	Descriptive information about message producer The desired producer is designated by its producer identifier (<i>producerID</i>). The value returned is a JMX <code>CompositeData</code> object describing the producer; see Table 3–57 for lookup keys used with this object.
getProducerInfo	None	CompositeData[]	Descriptive information about all current message producers The value returned is an array of JMX <code>CompositeData</code> objects describing the producers; see Table 3–57 for lookup keys used with these objects.
getProducerWildcards	None	String[]	Wildcard strings used by current producers associated with the broker
getNumWildcardProducers	wildcard-String	Integer	Number of current producers associated with the broker that are using the specified wildcard string

The `getProducerInfoByID` and `getProducerInfo` operations return objects implementing the JMX interface `CompositeData`, 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	Value Type	Description
<code>ProducerID</code>	String	Producer identifier
<code>ServiceName</code>	String	Name of associated connection service
<code>ConnectionID</code>	String	Connection identifier of associated connection
<code>Host</code>	String	Connection's host name
<code>User</code>	String	Connection's user name
<code>DestinationName</code>	String	Name of associated destination

TABLE 3–57 Lookup Keys for Message Producer Information (Continued)

Name	Value Type	Description
DestinationNames	String[]	Destination names that match wildcards used by wildcard producers For topic destinations only.
Wildcard	Boolean	Wildcard producer? For topic destinations only.
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
<code>NumConsumers</code>	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
<code>getConsumerIDs</code>	None	<code>String[]</code>	Consumer identifiers of all current message consumers
<code>purge</code> ¹	<code>consumerID (String)</code>	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
NumWildcardConsumers	Integer	No	Number of wildcard message consumers associated with the broker

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

TABLE 3-62 Consumer Manager Monitor Operations *(Continued)*

Name	Parameters	Result Type	Description
getConsumerWildcards	none	String[]	Wildcard strings used by current consumers associated with the broker
getNumWildcardConsumers	wildcard-String	Integer	Number of current consumers associated with the broker that are using the specified wildcard string

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
DestinationNames	String[]	Destination names that match wildcards used by wildcard consumers For topic destinations only.
Wildcard	Boolean	Wildcard consumer? For topic destinations only.
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.

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

Name	Value Type	Description
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
ClientID ¹	String	Client identifier
DurableActive ¹	Boolean	Subscriber active?
FlowPaused	Boolean	Message delivery paused?
NumMsgs	Long	Cumulative number of messages that have been dispatched to consumer (includes messages that have been delivered and those waiting to be delivered)
NumMsgsPending	Long	Current number of messages that have been dispatched to consumer and are being held in broker memory and persistent store (includes messages that have been delivered and those waiting to be delivered)
NumMsgsPendingAcks	Long	Current number of messages that have been delivered to consumer and are being held in broker memory and persistent store pending acknowledgment
NextMessageID	Long	JMS Message ID of the next message to be delivered to consumer
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
TransactionID	String	Transaction identifier
XID ¹	String	Distributed transaction identifier (XID)
User	String	User name
ClientID	String	Client identifier
ConnectionString	String	Connection string
CreationTime	Long	Time created, in standard Java format (milliseconds since January 1, 1970, 00:00:00 UTC)
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 (jconsole). See Table 3–71 for possible values.
NumMsgs	Long	Number of messages
NumAcks	Long	Number of acknowledgments

¹ Distributed transactions only

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	TransactionState.CREATED	CREATED	Transaction created
1	TransactionState.STARTED	STARTED	Transaction started
2	TransactionState.FAILED	FAILED	Transaction has failed
3	TransactionState.INCOMPLETE	INCOMPLETE	Transaction incomplete
4	TransactionState.COMPLETE	COMPLETE	Transaction complete
5	TransactionState.PREPARED	PREPARED	Transaction in prepared state ¹
6	TransactionState.COMMITTED	COMMITTED	Transaction committed

¹ Distributed transactions only

TABLE 3–71 Transaction State Values (Continued)

Value	Utility Constant	String Representation	Meaning
7	TransactionState.ROLLEDBACK	ROLLEDBACK	Transaction rolled back
8	TransactionState.TIMED_OUT	TIMED_OUT	Transaction has timed out
–1	TransactionState.UNKNOWN	UNKNOWN	Transaction state unknown

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
mq.transaction.prepare ¹	TransactionNotification.TRANSACTION_PREPARE	Transaction has entered prepared state
mq.transaction.commit	TransactionNotification.TRANSACTION_COMMIT	Transaction committed
mq.transaction.rollback	TransactionNotification.TRANSACTION_ROLLBACK	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
getTransactionID	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	Cluster identifier Must be a unique alphanumeric string of no more than $n - 13$ characters, where n is the maximum table name length allowed by the database. No two running clusters may have the same cluster identifier. This string is appended to the names of all database tables in the cluster's shared persistent store. Note – For brokers belonging to an HA cluster, this attribute is used in database table names in place of <code>BrokerID</code> (see Table 3–1).
ConfigFileURL ²	String	Yes	URL of cluster configuration file
LocalBrokerInfo	CompositeData	No	Descriptive information about local broker The value returned is a <code>JMX CompositeData</code> object describing the broker; see Table 3–76 for lookup keys used with this object.
MasterBrokerInfo ²	CompositeData	No	Descriptive information about master broker The value returned is a <code>JMX CompositeData</code> object describing the master broker; see Table 3–76 for lookup keys used with this object.
UseSharedDatabaseForConfigRecord ²	Boolean	No	Does conventional cluster use a shared JDBC data store instead of a master broker for the cluster configuration change record?

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
<code>getBrokerAddresses</code>	None	<code>String[]</code>	<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:</p> <pre>host1:3000</pre> <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>
<code>getBrokerIDs¹</code>	None	<code>String[]</code>	<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>
<code>getBrokerInfoByAddress</code>	<i>brokerAddress</i> (String)	<code>CompositeData</code>	<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>
<code>getBrokerInfoByID¹</code>	<i>brokerID</i> (String)	<code>CompositeData</code>	<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>

¹ HA clusters only

TABLE 3-75 Cluster Configuration Operations (Continued)

Name	Parameters	Result Type	Description
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>
reload ²	None	None	Reload cluster configuration file
changeMasterBroker ²	<i>oldMasterBroker</i> (String), <i>newMasterBroker</i> (String)	CompositeData	<p>Specify a change of master broker from <i>oldMasterBroker</i> to <i>newMasterBroker</i>, where both arguments are in <code>imq.cluster.masterbroker</code> format (host:port).</p> <p>The value returned is a JMX <code>CompositeData</code> object containing information about the success or failure of the operation; see Table 3-77 for lookup keys used with this object.</p> <p>This operation can only be performed on the broker that is the current master broker. If it is performed on any other broker it will have no effect and the <code>CompositeData</code> object returned will contain details of the error.</p> <p>This operation must not be performed on a broker whose lifecycle is being managed by GlassFish Server. In this case GlassFish Server tools must be used instead.</p>

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

The `changeMasterBroker` operation returns an object implementing the JMX interface `CompositeData`, which maps lookup keys to associated data values. The keys shown in [Table 3–77](#) are defined as static constants in the utility class `ChangeMasterBrokerResultInfo` for use with this object.

TABLE 3–77 Lookup Keys for `changeMasterBroker`

Key	Value Type	Description
Success	Boolean	Whether an error occurred when performing the <code>changeMasterBroker</code> operation. If an error occurred, the <code>StatusCode</code> and <code>DetailMessage</code> keys contain more information.
StatusCode	Integer	A status code set when an error occurred. The <code>DetailMessage</code> key contains more information.
DetailMessage	String	An error message set when an error occurs. The possible errors, and the actions that should be taken to resolve them, are the same as for the <code>imqcmd changemaster</code> command, as described in “To Change the Master Broker Dynamically While the Cluster Is Running” in <i>Open Message Queue 4.5 Administration Guide</i> .

Notification

The cluster configuration MBean supports the notification shown in [Table 3–78](#).

TABLE 3–78 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 3–79](#). The names of these attributes are defined as static constants in the utility class `ClusterAttributes`.

TABLE 3–79 Cluster Monitor Attributes

Name	Type	Settable?	Description
HighlyAvailable	Boolean	No	High-availability (HA) cluster?
ClusterID ¹	String	No	Cluster identifier Must be a unique alphanumeric string of no more than $n - 13$ characters, where n is the maximum table name length allowed by the database. No two running clusters may have the same cluster identifier. This string is appended to the names of all database tables in the cluster's shared persistent store. Note – For brokers belonging to an HA cluster, this attribute is used in database table names in place of <code>BrokerID</code> (see Table 3–4).
ConfigFileURL ²	String	Yes	URL of cluster configuration file
LocalBrokerInfo	CompositeData	No	Descriptive information about local broker The value returned is a <code>JMX CompositeData</code> object describing the broker; see Table 3–81 for lookup keys used with this object.
MasterBrokerInfo ²	CompositeData	No	Descriptive information about master broker The value returned is a <code>JMX CompositeData</code> object describing the master broker; see Table 3–81 for lookup keys used with this object.
UseSharedDatabaseForConfigRecord ²	Boolean	No	Does conventional cluster use a shared JDBC data store instead of a master broker for the cluster configuration change record?

¹ HA clusters only

² Conventional clusters only

Operations

The cluster monitor MBean supports the operations shown in [Table 3–80](#). The names of these operations are defined as static constants in the utility class `ClusterOperations`.

TABLE 3–80 Cluster Monitor Operations

Name	Parameters	Result Type	Description
<code>getBrokerAddresses</code>	None	<code>String[]</code>	<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:</p> <pre>host1:3000</pre> <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>
<code>getBrokerIDs¹</code>	None	<code>String[]</code>	<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>
<code>getBrokerInfoByAddress</code>	<i>brokerAddress</i> (String)	<code>CompositeData</code>	<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 3–81 for lookup keys used with this object.</p>
<code>getBrokerInfoByID¹</code>	<i>brokerID</i> (String)	<code>CompositeData</code>	<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 3–81 for lookup keys used with this object. For conventional clusters, the operation returns <code>null</code>.</p>

¹ HA clusters only

TABLE 3–80 Cluster Monitor Operations (Continued)

Name	Parameters	Result Type	Description
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 3–81 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>

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 3–81](#) are defined as static constants in the utility class `BrokerClusterInfo` for use with these objects.

TABLE 3–81 Lookup Keys for Cluster Monitor Information

Key	Value Type	Description
Address	String	<p>Broker address, in the form <i>hostName:portNumber</i></p> <p>Example: <code>host1:3000</code></p>
ID ¹	String	Broker identifier
State	Integer	<p>Current state of broker</p> <p>See Table 3–82 for possible values.</p>
StateLabel	String	<p>String representation of current broker state</p> <p>Useful for displaying the state in human-readable form, such as in the Java Monitoring and Management Console (<code>jconsole</code>).</p> <p>See Table 3–82 for possible values.</p>
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

¹ HA clusters only

TABLE 3–81 Lookup Keys for Cluster Monitor Information (Continued)

Key	Value Type	Description
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>mq.cluster.monitor.interval</code> .

¹ HA clusters only

Table 3–82 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 3–82 Broker State Values

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

Notifications

The cluster monitor MBean supports the notifications shown in Table 3–83. 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 3–83 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

TABLE 3–83 Cluster Monitor Notifications (Continued)

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

¹ HA clusters only

Table 3–84 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 3–84 Data Retrieval Methods for Cluster Monitor Notifications

Method	Result Type	Description
isHighlyAvailable	Boolean	High-availability (HA) cluster?
getClusterID	String	Cluster identifier
getBrokerID	String	Broker identifier of affected broker
getBrokerAddress	String	Address of affected broker, in the form <i>hostName:portNumber</i> Example: host1:3000
isMasterBroker 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 3–85](#). The names of these attributes are defined as static constants in the utility class `LogAttributes`.

TABLE 3–85 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 3–86 for possible values.
RolloverBytes	Long	Yes	File length, in bytes, at which output rolls over to a new log file A value of <code>-1</code> 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 <code>-1</code> denotes an unlimited number of seconds (no rollover based on file age).

[Table 3–86](#) 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 3–86 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 3–87](#).

TABLE 3–87 Log Configuration Notification

Name	Description
jmx.attribute.change	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 3–88](#). 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 3–88 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 3–89](#) shows the methods defined in class `LogNotification` for obtaining details about a log monitor notification.

TABLE 3–89 Data Retrieval Methods for Log Monitor Notifications

Method	Result Type	Description
getLevel	String	Logging level of logged message See Table 3–86 for possible values.
getMessage	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 3–90](#). The names of these attributes are defined as static constants in the utility class `JVMAttributes`.

TABLE 3–90 JVM Monitor Attributes

Name	Type	Settable?	Description
TotalMemory	Long	No	Current total memory, in bytes

TABLE 3-90 JVM Monitor Attributes (Continued)

Name	Type	Settable?	Description
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 OutOfMemoryError 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	Table 3–1
	Broker Monitor	Table 3–4
ClientID	Connection Monitor	Table 3–24
ClientPlatform	Connection Monitor	Table 3–24
ClusterID	Cluster Configuration	Table 3–74
	Cluster Monitor	Table 3–79

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

Attribute	MBean	Reference
ConfigFileURL	Cluster Configuration	Table 3-74
	Cluster Monitor	Table 3-79
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 3-90
HighlyAvailable	Cluster Configuration	Table 3-74
	Cluster Monitor	Table 3-79
Host	Connection Monitor	Table 3-24
InitMemory	JVM Monitor	Table 3-90
InstanceName	Broker Configuration	Table 3-1
	Broker Monitor	Table 3-4
Level	Log Configuration	Table 3-85
LimitBehavior	Destination Configuration	Table 3-32
LocalBrokerInfo	Cluster Configuration	Table 3-74
	Cluster Monitor	Table 3-79
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 3-79

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 3-90
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
NextMessageID	Destination Monitor	Table 3-38
NumActiveConsumers	Destination Monitor	Table 3-38
NumActiveThreads	Service Manager Monitor	Table 3-19
	Service Monitor	Table 3-12

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

Attribute	MBean	Reference
NumBackupConsumers	Destination Monitor	Table 3-38
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
NumMsgsRemote	Destination Monitor	Table 3-38
NumPktsIn	Service Manager Monitor	Table 3-19
	Service Monitor	Table 3-12

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

Attribute	MBean	Reference
NumPktsOut	Service Manager Monitor	Table 3-19
	Service Monitor	Table 3-12
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
NumWildcards	Destination Monitor	Table 3-38
NumWildcardConsumers	Consumer Manager Monitor	Table 3-61
	Destination Monitor	Table 3-38
NumWildcardProducers	Producer Manager Monitor	Table 3-55
	Destination Monitor	Table 3-38
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

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

Attribute	MBean	Reference
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
ResourceState	Broker Monitor	Table 3-4
ReloadXMLSchemaOn Failure	Destination Configuration	Table 3-32
ResourceState	Broker Monitor	Table 3-4
RolloverBytes	Log Configuration	Table 3-85
RolloverSecs	Log Configuration	Table 3-85
ServiceName	Connection Monitor	Table 3-24
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 3-90
TotalMsgBytes	Destination Manager Monitor	Table 3-49
	Destination Monitor	Table 3-38
TotalMsgBytesRemote	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
ValidateXMLSchemaEnabled	Destination Configuration	Table 3-32

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

Attribute	MBean	Reference
Version	Broker Configuration	Table 3-1
	Broker Monitor	Table 3-4
XMLSchemaURLList	Destination Configuration	Table 3-32

[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
getActiveConsumerIDs	Destination Monitor	Table 3-41
getBackupConsumerIDs	Destination Monitor	Table 3-41
getBrokerAddresses	Cluster Configuration	Table 3-75
	Cluster Monitor	Table 3-80
getBrokerIDs	Cluster Configuration	Table 3-75
	Cluster Monitor	Table 3-80
getBrokerInfo	Cluster Configuration	Table 3-75
	Cluster Monitor	Table 3-80
getBrokerInfoByAddress	Cluster Configuration	Table 3-75
	Cluster Monitor	Table 3-80
getBrokerInfoByID	Cluster Configuration	Table 3-75
	Cluster Monitor	Table 3-80
getConnection	Destination Monitor	Table 3-41

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

Operation	MBean	Reference
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
getConsumerWildcards	Consumer Manager Monitor	Table 3-62
	Destination Monitor	Table 3-41
getDestinations	Destination Manager Configuration	Table 3-45
	Destination Manager Monitor	Table 3-50
getNumWildcardConsumers	Consumer Manager Monitor	Table 3-62
	Destination Monitor	Table 3-41
getNumWildcardProducers	Producer Manager Monitor	Table 3-56
	Destination Monitor	Table 3-41
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
getProducerWildcards	Destination Monitor	Table 3-41
	Producer Manager	Table 3-56
getProperty	Broker Configuration	Table 3-2
getService	Connection Monitor	Table 3-25

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

Operation	MBean	Reference
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
getWildcards	Destination Monitor	Table 3-41
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
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 3-78
	Destination Configuration	Table 3-37
	Destination Manager Configuration	Table 3-48
	Log Configuration	Table 3-87
	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.resource.state.change	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 3-83
mq.broker.takeover.fail	Broker Monitor	Table 3-5
	Cluster Monitor	Table 3-83
mq.broker.takeover.start	Broker Monitor	Table 3-5
	Cluster Monitor	Table 3-83
mq.cluster.broker.down	Cluster Monitor	Table 3-83
mq.cluster.broker.join	Broker Monitor	Table 3-5
	Cluster Monitor	Table 3-83
mq.connection.close	Connection Manager Monitor	Table 3-30
	Service Monitor	Table 3-15
mq.connection.open	Connection Manager Monitor	Table 3-30
	Service Monitor	Table 3-15
mq.connection.reject	Connection Manager Monitor	Table 3-30
	Service Monitor	Table 3-15

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

Notification	MBean	Reference
mq.destination.compact	Destination Manager Monitor	Table 3-51
	Destination Monitor	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	Table 3-51
	Destination Monitor	Table 3-42
mq.destination.purge	Destination Manager Monitor	Table 3-51
	Destination Monitor	Table 3-42
mq.destination.resume	Destination Manager Monitor	Table 3-51
	Destination Monitor	Table 3-42
mq.log.level.ERROR	Log Monitor	Table 3-88
mq.log.level.INFO	Log Monitor	Table 3-88
mq.log.level.WARNING	Log Monitor	Table 3-88
mq.service.pause	Service Manager Monitor	Table 3-21
	Service Monitor	Table 3-15
mq.service.resume	Service Manager Monitor	Table 3-21
	Service Monitor	Table 3-15
mq.transaction.commit	Transaction Manager Monitor	Table 3-72
mq.transaction.prepare	Transaction Manager Monitor	Table 3-72
mq.transaction.rollback	Transaction Manager Monitor	Table 3-72

Index

A

- AcknowledgeMode lookup key
 - message consumer, 93, 95
- AcknowledgeModeLabel lookup key
 - message consumer, 94, 95
- acknowledgment modes, 93, 94
 - table, 95
- addNotificationListener method, interface
 - MBeanServerConnection, 47
- Address lookup key
 - broker cluster, 103, 106
- admin connection factory
 - configuring, 32
 - defined, 31
 - obtaining JMX connector from, 32–33
 - obtaining JMX connector without, 33–34
- admin connection service name, 25
- AdminConnectionConfiguration utility class, 29, 32
- AdminConnectionFactory class, 29, 31, 32
- ALL utility constant
 - class DestinationPauseType, 75, 84
- Attribute class, 35
- attribute lists, defined, 35
- AttributeList class, 35
- attributes, MBean
 - accessing, 34–39
 - alphabetical list (table), 113–119
 - broker configuration MBean, 52
 - broker monitor MBean, 55–56
 - cluster configuration MBean, 100–101
 - cluster monitor MBean, 104–105
 - combining with operations, 42
 - attributes, MBean (*Continued*)
 - connection configuration MBean, 66–67
 - connection manager configuration MBean, 68–69
 - connection manager monitor MBean, 69–70
 - connection monitor MBean, 67–68
 - consumer manager configuration MBean, 91
 - consumer manager monitor MBean, 92
 - defined, 20
 - destination configuration MBean, 71–74
 - destination manager configuration MBean, 82–83
 - destination manager monitor MBean, 85–86
 - destination monitor MBean, 76–79
 - JVM monitor MBean, 111–112
 - log configuration MBean, 109–110
 - producer manager configuration MBean, 87–88
 - producer manager monitor MBean, 88
 - service configuration MBean, 59–60
 - service manager configuration MBean, 63–64
 - service manager monitor MBean, 64–65
 - service monitor MBean, 61–62
 - transaction manager configuration MBean, 96
 - transaction manager monitor MBean, 97
 - authentication credentials, 34
 - AUTO_ACKNOWLEDGE utility constant, interface
 - Session, 95
 - AutoCreateQueueMaxNumActiveConsumers attribute,
 - destination manager configuration MBean, 82
 - AutoCreateQueueMaxNumBackupConsumers attribute,
 - destination manager configuration MBean, 82
 - AutoCreateQueues attribute, destination manager configuration MBean, 82

AutoCreateTopics attribute, destination manager configuration MBean, 82
AvgNumActiveConsumers attribute
 destination monitor MBean, 54, 77
AvgNumBackupConsumers attribute
 destination monitor MBean, 54, 77
AvgNumConsumers attribute
 destination monitor MBean, 54, 77
AvgNumMsgs attribute
 destination monitor MBean, 54, 78
AvgTotalMsgBytes attribute
 destination monitor MBean, 54, 79

B

Broker MBean type, 23
broker clusters, 99–108
 cluster identifier, 100, 104, 108
 composite data object, lookup keys for (table), 103
 configuration MBean, 99–103
 high-availability (HA), 53, 100, 101, 102, 103, 104, 105, 106, 107, 108
 monitor MBean, 103–108
BROKER_CONFIG_MBEAN_NAME utility constant
 class MQObjectName, 26, 52
broker configuration MBean, 51–55
 attributes, 52
 notification, 55
 object name, 52
 operations, 52–55
BROKER_DOWN utility constant, class BrokerState, 107
broker monitor MBean, 55–58
 attributes, 55–56
 notification objects, 57
 notifications, 56–58
 object name, 55
BROKER_MONITOR_MBEAN_NAME utility constant
 class MQObjectName, 26, 55
BROKER_QUIESCE_COMPLETE utility constant, class BrokerNotification, 57
BROKER_QUIESCE_START utility constant, class BrokerNotification, 57
BROKER_SHUTDOWN_START utility constant, class BrokerNotification, 57
BROKER_TAKEOVER_COMPLETE utility constant
 class BrokerNotification, 57, 108
BROKER_STATE_CHANGE utility constant, class BrokerNotification, 57
BROKER_TAKEOVER_FAIL utility constant
 class BrokerNotification, 57, 108
BROKER_TAKEOVER_START utility constant
 class BrokerNotification, 57, 108
BrokerAttributes utility class, 30, 52, 55
BrokerClusterInfo utility class, 31, 102, 106
BrokerID attribute
 broker configuration MBean, 52
 broker monitor MBean, 56
BrokerNotification class, 30
 BROKER_QUIESCE_COMPLETE constant, 57
 BROKER_QUIESCE_START constant, 57
 BROKER_SHUTDOWN_START constant, 57
 BROKER_TAKEOVER_COMPLETE constant, 57, 108
 BROKER_STATE_CHANGE constant, 57
 BROKER_TAKEOVER_FAIL constant, 57, 108
 BROKER_TAKEOVER_START constant, 57, 108
 data retrieval methods, 57
 utility constants, 56, 107
BrokerOperations utility class, 30, 52
brokers, 51–58
 See brokers
 broker identifier, 52, 53, 56, 57, 101, 103, 105, 106, 108
 configuration MBean, 51–55
 monitor MBean, 55–58
 state values (table), 107
BrokerState utility class, 30, 107
 BROKER_DOWN constant, 107
 OPERATING constant, 107
 QUIESCE_COMPLETE constant, 107
 QUIESCE_STARTED constant, 107
 SHUTDOWN_STARTED constant, 107
 TAKEOVER_COMPLETE constant, 107
 TAKEOVER_FAILED constant, 107
 TAKEOVER_STARTED constant, 107
 UNKNOWN constant, 107

C

changeMasterBroker operation, cluster configuration MBean, 102

classes

- AdminConnectionConfiguration, 29, 32
- AdminConnectionFactory, 29, 31, 32
- Attribute, 35
- AttributeList, 35
- BrokerAttributes, 30, 52, 55
- BrokerClusterInfo, 31, 102, 106
- BrokerNotification, 30, 56, 57, 107
- BrokerOperations, 30, 52
- BrokerState, 30, 107
- ClusterAttributes, 31, 100, 104
- ClusterNotification, 31, 56, 107, 108
- ClusterOperations, 31, 101, 105
- ConnectionAttributes, 30, 66, 67, 68, 69
- ConnectionNotification, 30, 62, 70
- ConnectionOperations, 30, 68, 69, 70
- ConsumerAttributes, 31, 91, 92
- ConsumerInfo, 31, 93
- ConsumerOperations, 31, 91, 92
- DestinationAttributes, 30, 71, 76, 82, 83, 85
- DestinationLimitBehavior, 30, 74
- DestinationNotification, 30, 81, 86
- DestinationOperations, 30, 74, 79, 83, 86
- DestinationPauseType, 30, 75, 84
- DestinationState, 30, 79
- DestinationType, 24, 30, 73, 79, 84, 90, 94
- JMXConnectorFactory, 33
- JVMAttributes, 31, 111
- LogAttributes, 31, 109
- LogLevel, 31, 109
- LogNotification, 31, 110
- MQNotification, 30
- MQObjectName, 20, 25, 30
- Notification, 47
- NotificationFilterSupport, 48
- ObjectName, 20, 23
- ProducerAttributes, 30, 87, 88
- ProducerInfo, 31, 89
- ProducerOperations, 30, 88
- ServiceAttributes, 30, 59, 61, 63, 64
- ServiceNotification, 30, 62, 63, 65, 66

classes (*Continued*)

- ServiceOperations, 30, 60, 62, 64, 65
- ServiceState, 30, 62
- TransactionAttributes, 31, 96, 97
- TransactionInfo, 31, 97
- TransactionNotification, 31, 99
- TransactionOperations, 31, 96, 97
- TransactionState, 31, 98
- utility, 30–31
- CLIENT_ACKNOWLEDGE utility constant, interface Session, 95
- ClientID attribute, connection monitor MBean, 67
- ClientID lookup key
 - message consumer, 94
 - transaction, 98
- ClientPlatform attribute, connection monitor MBean, 67
- Cluster MBean type, 24
- CLUSTER_BROKER_DOWN utility constant, class ClusterNotification, 108
- CLUSTER_BROKER_JOIN utility constant
 - class ClusterNotification, 57, 107
- CLUSTER_CONFIG_MBEAN_NAME utility constant
 - class MQObjectName, 26, 100
- cluster configuration MBean, 99–103
 - attributes, 100–101
 - notification, 103
 - object name, 100
 - operations, 101–103
- cluster monitor MBean, 103–108
 - attributes, 104–105
 - notification objects, 108
 - notifications, 107–108
 - object name, 104
 - operations, 105–107
- CLUSTER_MONITOR_MBEAN_NAME utility constant
 - class MQObjectName, 26, 104
- ClusterAttributes utility class, 31, 100, 104
- ClusterID attribute
 - cluster configuration MBean, 100
 - cluster monitor MBean, 104
- ClusterNotification class, 31
 - CLUSTER_BROKER_DOWN constant, 108
 - CLUSTER_BROKER_JOIN constant, 57, 107

- ClusterNotification class (*Continued*)
 - data retrieval methods, 108
 - utility constants, 56, 107
- ClusterOperations utility class, 31, 101, 105
- clusters, *See* broker clusters
- com.sun.messaging package, 29, 32
- com.sun.messaging.jms.management.server
 - package, 29
- commit operation, transaction manager configuration MBean, 96
- COMMITTED utility constant, class TransactionState, 98
- compact operation
 - destination configuration MBean, 75
 - destination manager configuration MBean, 84
- COMPLETE utility constant, class TransactionState, 98
- composite data objects
 - See also* lookup keys
 - for broker clusters, 101, 102, 103, 105, 106
 - defined, 44
 - for brokers, 100, 104
 - for message consumers, 92, 93–94
 - for message producers, 89
 - for transactions, 97, 98
- CompositeData interface, 44, 89, 92, 93, 97, 100, 101, 102, 104, 105, 106
- Config MBean subtype, 24
- ConfigFileURL attribute
 - cluster configuration MBean, 100
 - cluster monitor MBean, 104
- configuration MBeans, 21
- connect method
 - class JMXConnectorFactory, 33, 34
- Connection MBean type, 24
- CONNECTION_CLOSE utility constant
 - class ConnectionNotification, 63, 70
- connection configuration MBean, 66–67
 - attribute, 66–67
 - object name, 66
- CONNECTION_MANAGER_CONFIG_MBEAN_NAME utility constant
 - class MQObjectName, 26, 68
- connection manager configuration MBean, 68–69
 - attribute, 68–69
- connection manager configuration MBean (*Continued*)
 - object name, 68
 - operations, 69
- connection manager monitor MBean, 69–70
 - attributes, 69–70
 - notification objects, 70
 - notifications, 70
 - object name, 69
 - operation, 70
- CONNECTION_MANAGER_MONITOR_MBEAN_NAME utility constant
 - class MQObjectName, 26, 69
- connection monitor MBean, 67–68
 - attributes, 67–68
 - object name, 67
 - operations, 68
- CONNECTION_OPEN utility constant
 - class ConnectionNotification, 63, 70
- CONNECTION_REJECT utility constant
 - class ConnectionNotification, 63, 70
- connection services, 58–66
 - configuration MBean, 58–60
 - manager configuration MBean, 63–64
 - manager monitor MBean, 64–66
 - monitor MBean, 60–63
 - names (table), 25
 - state values (table), 62
- ConnectionAttributes utility class, 30, 66, 67, 68, 69
- ConnectionID attribute
 - connection configuration MBean, 67
 - connection monitor MBean, 67
 - destination monitor MBean, 76
- ConnectionID lookup key
 - message consumer, 93
 - message producer, 89
- ConnectionFactory MBean type, 24
- ConnectionNotification class, 30
 - CONNECTION_CLOSE constant, 63, 70
 - CONNECTION_OPEN constant, 63, 70
 - CONNECTION_REJECT constant, 63, 70
 - data retrieval methods, 70
 - utility constants, 62, 70
- ConnectionOperations utility class, 30, 68, 69, 70
- connections, 66–70

- connections (*Continued*)
 - configuration MBean, 66–67
 - connection identifier, 66, 67, 69, 76, 89, 93
 - manager configuration MBean, 68–69
 - manager monitor MBean, 69–70
 - monitor MBean, 67–68
 - ConnectionString lookup key, transaction, 98
 - connectors, JMX, *See* JMX connectors
 - CONSUMER_MANAGER_CONFIG_MBEAN_NAME utility
 - constant
 - class MQObjectName, 26, 91
 - consumer manager configuration MBean, 90–91
 - attribute, 91
 - object name, 91
 - operations, 91
 - consumer manager monitor MBean, 91–95
 - attribute, 92
 - object name, 92
 - operations, 92–95
 - CONSUMER_MANAGER_MONITOR_MBEAN_NAME utility
 - constant
 - class MQObjectName, 26, 92
 - ConsumerAttributes utility class, 31, 91, 92
 - ConsumerFlowLimit attribute, destination
 - configuration MBean, 72
 - ConsumerID lookup key, message consumer, 93
 - ConsumerInfo utility class, 31, 93
 - ConsumerManager MBean type, 24
 - ConsumerOperations utility class, 31, 91, 92
 - consumers, *See* message consumers
 - CONSUMERS utility constant
 - class DestinationPauseType, 75, 84
 - CONSUMERS_PAUSED utility constant, class
 - DestinationState, 79
 - create operation
 - destination manager configuration MBean, 83, 84
 - createConnection method, class
 - AdminConnectionFactory, 32
 - createConnectionConfig utility method
 - class MQObjectName, 26, 66
 - createConnectionMonitor utility method
 - class MQObjectName, 26, 67
 - CREATED utility constant, class TransactionState, 98
 - CreatedByAdmin attribute, destination monitor
 - MBean, 76
 - createDestinationConfig utility method
 - class MQObjectName, 26, 71
 - createDestinationMonitor utility method
 - class MQObjectName, 26, 76
 - createServiceConfig utility method
 - class MQObjectName, 26, 59
 - createServiceMonitor utility method
 - class MQObjectName, 26, 60
 - CreationTime lookup key, transaction, 98
 - CREDENTIALS attribute
 - (JMXConnectorFactory.connect environment), 34
- ## D
- dead message queue, 72, 73, 82, 83, 85
 - Destination MBean type, 24
 - DESTINATION_COMPACT utility constant
 - class DestinationNotification, 81, 86
 - destination configuration MBean, 71–75
 - attributes, 71–74
 - notification, 75
 - object name, 71
 - operations, 74–75
 - DESTINATION_CREATE utility constant, class
 - DestinationNotification, 86
 - DESTINATION_DESTROY utility constant, class
 - DestinationNotification, 86
 - DESTINATION_MANAGER_CONFIG_MBEAN_NAME utility
 - constant
 - class MQObjectName, 26, 82
 - destination manager configuration MBean, 81–85
 - attributes, 82–83
 - notification, 84–85
 - object name, 82
 - operations, 83–84
 - destination manager monitor MBean, 85–87
 - attributes, 85–86
 - notification objects, 86
 - notifications, 86–87
 - object name, 85
 - operation, 86

- DESTINATION_MANAGER_MONITOR_MBEAN_NAME utility
 - constant
 - class MQObjectName, 26, 85
 - destination monitor MBean, 75–81
 - attributes, 76–79
 - notification objects, 81
 - notifications, 81
 - object name, 75–76
 - operations, 79–81
 - DESTINATION_PAUSE utility constant
 - class DestinationNotification, 81, 86
 - DESTINATION_PURGE utility constant
 - class DestinationNotification, 81, 86
 - DESTINATION_RESUME utility constant
 - class DestinationNotification, 81, 86
 - destination types (table), 24
 - DestinationAttributes utility class, 30, 71, 76, 82, 83, 85
 - DestinationLimitBehavior utility class, 30, 74
 - FLOW_CONTROL constant, 74
 - REJECT_NEWEST constant, 74
 - REMOVE_LOW_PRIORITY constant, 74
 - REMOVE_OLDEST constant, 74
 - DestinationManager MBean type, 24
 - DestinationName lookup key
 - message consumer, 93
 - message producer, 89
 - DestinationNames lookup key
 - message consumer, 93
 - message producer, 90
 - DestinationNotification class, 30
 - data retrieval methods, 81, 86
 - DESTINATION_COMPACT constant, 81, 86
 - DESTINATION_CREATE constant, 86
 - DESTINATION_DESTROY constant, 86
 - DESTINATION_PAUSE constant, 81, 86
 - DESTINATION_PURGE constant, 81, 86
 - DESTINATION_RESUME constant, 81, 86
 - utility constants, 81, 86
 - DestinationOperations utility class, 30, 74, 79, 83, 86
 - DestinationPauseType utility class, 30, 75, 84
 - ALL constant, 75, 84
 - CONSUMERS constant, 75, 84
 - PRODUCERS constant, 75, 84
 - destinations, 71–87
 - configuration MBean, 71–75
 - limit behavior (table), 74
 - manager configuration MBean, 81–85
 - manager monitor MBean, 85–87
 - monitor MBean, 75–81
 - pause types (table), 75
 - types (table), 24
 - DestinationState utility class, 30, 79
 - CONSUMERS_PAUSED constant, 79
 - PAUSED constant, 79
 - PRODUCERS_PAUSED constant, 79
 - RUNNING constant, 79
 - UNKNOWN constant, 79
 - DestinationType lookup key
 - message consumer, 94
 - message producer, 90
 - DestinationType utility class, 24, 30, 73, 79, 84, 90, 94
 - destroy operation
 - connection manager configuration MBean, 69
 - destination manager configuration MBean, 84
 - dest type property (object name), 23
 - values (table), 24
 - DiskReserved attribute, destination monitor MBean, 79
 - DiskUsed attribute, destination monitor MBean, 79
 - DiskUtilizationRatio attribute, destination monitor MBean, 79
 - distributed transaction identifier (XID), 98
 - DMQTruncateBody attribute, destination manager configuration MBean, 82
 - DUPS_OK_ACKNOWLEDGE utility constant, interface Session, 95
 - Durable lookup key, message consumer, 94
 - DurableActive lookup key, message consumer, 94
 - DurableName lookup key, message consumer, 94
- E**
- Embedded attribute, broker monitor MBean, 56
 - enableType method, class
 - NotificationFilterSupport, 48
 - ERROR utility constant, class LogLevel, 109

F

FAILED utility constant, class TransactionState, 98
 FLOW_CONTROL utility constant, class
 DestinationLimitBehavior, 74
 FlowPaused lookup key
 message consumer, 94
 message producer, 90
 FreeMemory attribute, JVM monitor MBean, 112

G

getActiveConsumerIDs operation, destination monitor
 MBean, 80
 getAttribute method, interface
 MBeanServerConnection, 34
 getAttributes method, interface
 MBeanServerConnection, 35
 getBackupConsumerIDs operation, destination monitor
 MBean, 80
 getBrokerAddress method
 class BrokerNotification, 57
 class ClusterNotification, 108
 getBrokerAddresses operation
 cluster configuration MBean, 101
 cluster monitor MBean, 105
 getBrokerID method
 class BrokerNotification, 57
 class ClusterNotification, 108
 getBrokerIDs operation
 cluster configuration MBean, 101
 cluster monitor MBean, 105
 getBrokerInfo operation
 cluster configuration MBean, 102
 cluster monitor MBean, 106
 getBrokerInfoByAddress operation
 cluster configuration MBean, 101
 cluster monitor MBean, 105, 106
 getBrokerInfoByID operation
 cluster configuration MBean, 101, 102
 cluster monitor MBean, 105, 106
 getClusterID method, class
 ClusterNotification, 108
 getConnection operation, destination monitor
 MBean, 80

getConnectionID method, class
 ConnectionNotification, 70
 getConnections operation
 connection manager configuration MBean, 69
 connection manager monitor MBean, 70
 service monitor MBean, 62
 getConsumerIDs operation
 connection monitor MBean, 68
 consumer manager configuration MBean, 91
 consumer manager monitor MBean, 92
 destination monitor MBean, 80
 service monitor MBean, 62
 getConsumerInfo operation
 consumer manager monitor MBean, 92, 93
 getConsumerInfoByID operation
 consumer manager monitor MBean, 92, 93
 getConsumerWildcards operation
 consumer manager monitor MBean, 93
 destination monitor MBean, 80
 getCreatedByAdmin method
 class DestinationNotification, 81, 86
 getDestinationName method
 class DestinationNotification, 81, 86
 getDestinations operation
 destination manager configuration MBean, 83
 destination manager monitor MBean, 86
 getDestinationType method
 class DestinationNotification, 81, 86
 getFailedBrokerID method, class
 BrokerNotification, 57
 getHeapMemoryUsage method, class
 BrokerNotification, 58
 getLevel method, class LogNotification, 111
 getMBeanServerConnection method, class
 JMXConnector, 32
 getMessage method, class LogNotification, 111
 getName method, class Attribute, 35
 getNewResourceState method, class
 BrokerNotification, 58
 getNumWildcardConsumers operation
 consumer manager monitor MBean, 93
 destination monitor MBean, 80
 getNumWildcardProducers operation
 destination monitor MBean, 81

- getNumWildcardProducers operation (*Continued*)
 - producer manager monitor MBean, 89
- getOldResourceState method, class
 - BrokerNotification, 58
- getPauseType method
 - class DestinationNotification, 81, 87
- getProducerIDs operation
 - connection monitor MBean, 68
 - destination monitor MBean, 80
 - producer manager configuration MBean, 88
 - producer manager monitor MBean, 89
 - service monitor MBean, 62
- getProducerInfo operation
 - producer manager monitor MBean, 89
- getProducerInfoByID operation
 - producer manager monitor MBean, 89
- getProducerWildcards operation
 - destination monitor MBean, 80
 - producer manager monitor MBean, 89
- getRemoteHost method, class
 - ConnectionNotification, 70
- getService operation, connection monitor MBean, 68
- getServiceName method
 - class ConnectionNotification, 70
 - class ServiceNotification, 63, 66
- getServices operation
 - service manager configuration MBean, 64
 - service manager monitor MBean, 65
- getTemporaryDestinations operation, connection monitor MBean, 68
- getTransactionID method, class
 - TransactionNotification, 99
- getTransactionIDs operation
 - transaction manager configuration MBean, 96
 - transaction manager monitor MBean, 97
- getTransactionInfo operation
 - transaction manager monitor MBean, 97
- getTransactionInfoByID operation
 - transaction manager monitor MBean, 97
- getUserName method, class
 - ConnectionNotification, 70
- getValue method, class Attribute, 35
- getWildcards operation, destination monitor MBean, 80

H

- HA, *See* high-availability broker clusters
- handback objects, 47
- handleNotification method, interface
 - NotificationListener, 47
- high-availability (HA) broker clusters
 - ClusterID attribute, cluster configuration MBean, 100
 - ClusterID attribute, cluster monitor MBean, 104
- getBrokerAddresses operation, cluster configuration MBean, 101
- getBrokerAddresses operation, cluster monitor MBean, 105
- getBrokerIDs operation, cluster monitor MBean, 105
- getBrokerInfo operation, cluster configuration MBean, 102
- getBrokerInfo operation, cluster monitor MBean, 106
- HighlyAvailable attribute, cluster configuration MBean, 100
- HighlyAvailable attribute, cluster monitor MBean, 104
- ID lookup key, composite data object, 103, 106
- isHighlyAvailable method, class
 - ClusterNotification, 108
- mq.broker.takeover.complete notification, cluster monitor MBean, 108
- mq.broker.takeover.fail notification, cluster monitor MBean, 108
- mq.broker.takeover.start notification, cluster monitor MBean, 108
- NumMsgs lookup key, composite data object, 106
- shutdown operation, broker configuration MBean, 53
- StatusTimestamp lookup key, composite data object, 107
- TAKEOVER_COMPLETE state, 107
- TAKEOVER_FAILED state, 107
- TAKEOVER_STARTED state, 107
- TakeoverBrokerID lookup key, composite data object, 106
- HighlyAvailable attribute
 - cluster configuration MBean, 100

HighlyAvailable attribute (*Continued*)
 cluster monitor MBean, 104
 Host attribute, connection monitor MBean, 67
 Host lookup key
 message consumer, 93
 message producer, 89
 HTTP, *See* Hypertext Transfer Protocol
 httpjms connection service name, 25
 HTTPS, *See* Hypertext Transfer Protocol, Secure
 httpsjms connection service name, 25
 Hypertext Transfer Protocol (HTTP), 25
 Hypertext Transfer Protocol, Secure (HTTPS), 25

I

ID lookup key
 broker cluster, 103, 106
 id property (object name), 23
 INCOMPLETE utility constant, class
 TransactionState, 98
 INFO utility constant, class LogLevel, 109
 InitMemory attribute, JVM monitor MBean, 112
 InstanceName attribute
 broker configuration MBean, 52
 broker monitor MBean, 56
 interfaces
 CompositeData, 44, 89, 92, 93, 97, 100, 101, 102,
 104, 105, 106
 NotificationListener, 47
 Session, 95
 invoke method
 interface MBeanServerConnection, 39, 40
 isHighlyAvailable method, class
 ClusterNotification, 108
 isMasterBroker method, class
 ClusterNotification, 108

J

Java Management Extensions (JMX) Specification, 12
 Java Monitoring and Management Console
 (jconsole), 61, 76, 94, 98, 106

jconsole, *See* Java Monitoring and Management
 Console
 jms connection service name, 25
 jmx.attribute.change notification
 broker configuration MBean, 55
 cluster configuration MBean, 103
 destination configuration MBean, 75
 destination manager configuration MBean, 85
 log configuration MBean, 110
 service configuration MBean, 60
 JMX connectors
 defined, 20
 obtaining from admin connection factory, 32–33
 obtaining without admin connection factory, 33–34
 JMX service URLs, parameter to
 JMXConnectorFactory.connect method, 34
 JMXConnectorFactory class, 33
 JVM, *See* Java Virtual Machine
 JVM (Java Virtual Machine), 111–112
 monitor MBean, 111–112
 JVM MBean type, 24
 JVM monitor MBean, 21, 111–112
 attributes, 111–112
 object name, 111
 JVM_MONITOR_MBEAN_NAME utility constant
 class MQObjectName, 27, 111
 JVMAttributes utility class, 31, 111

L

LastAckTime lookup key, message consumer, 94
 Level attribute
 log configuration MBean, 109
 limit behavior, destinations, 74
 LimitBehavior attribute
 destination configuration MBean, 72, 74
 LocalBrokerInfo attribute
 cluster configuration MBean, 100, 102
 cluster monitor MBean, 104, 106
 LocalDeliveryPreferred attribute, destination
 configuration MBean, 73
 LocalOnly attribute, destination configuration
 MBean, 73
 Log MBean type, 24

- LOG_CONFIG_MBEAN_NAME utility constant
 - class MQObjectName, 27, 109
- log configuration MBean, 108–110
 - attributes, 109–110
 - notification, 110
 - object name, 109
- LOG_LEVEL_ERROR utility constant, class
 - LogNotification, 110
- LOG_LEVEL_INFO utility constant, class
 - LogNotification, 110
- LOG_LEVEL_WARNING utility constant, class
 - LogNotification, 110
- log monitor MBean, 110–111
 - notification objects, 110
 - notifications, 110–111
 - object name, 110
- LOG_MONITOR_MBEAN_NAME utility constant
 - class MQObjectName, 27, 110
- LogAttributes utility class, 31, 109
- LogDeadMsgs attribute, destination manager configuration MBean, 83
- logging, 108–111
 - configuration MBean, 108–110
 - monitor MBean, 110–111
- LogLevel utility class, 31, 109
 - ERROR constant, 109
 - INFO constant, 109
 - NONE constant, 109
 - UNKNOWN constant, 109
 - WARNING constant, 109
- LogNotification class, 31
 - data retrieval methods, 110
 - LOG_LEVEL_ERROR constant, 110
 - LOG_LEVEL_INFO constant, 110
 - LOG_LEVEL_WARNING constant, 110
 - utility constants, 110
- lookup keys
 - for broker clusters, 103
 - defined, 44
 - for message consumers, 93–94
 - for message producers, 89–90
 - for transactions, 98

M

- managed beans, *See* MBeans
- manager MBeans, 22
- MasterBrokerInfo attribute
 - cluster configuration MBean, 100
 - cluster monitor MBean, 104, 106
- MaxBytesPerMsg attribute
 - destination configuration MBean, 72
 - destination manager configuration MBean, 82
- MaxMemory attribute, JVM monitor MBean, 112
- MaxNumActiveConsumers attribute, destination configuration MBean, 72
- MaxNumBackupConsumers attribute, destination configuration MBean, 72
- MaxNumMsgs attribute
 - destination configuration MBean, 72
 - destination manager configuration MBean, 82
- MaxNumProducers attribute, destination configuration MBean, 72
- MaxThreads attribute
 - service configuration MBean, 59
 - service manager configuration MBean, 64
- MaxTotalMsgBytes attribute
 - destination configuration MBean, 72
 - destination manager configuration MBean, 82
- MBean server
 - connecting to, 31–34
 - connection, defined, 31
 - defined, 20
- MBeans
 - attributes, accessing, 34–39
 - broker configuration, 51–55
 - broker monitor, 55–58
 - cluster configuration, 99–103
 - cluster monitor, 103–108
 - combining operations and attributes, 42
 - configuration, defined, 21
 - connection configuration, 66–67
 - connection manager configuration, 68–69
 - connection manager monitor, 69–70
 - connection monitor, 67–68
 - consumer manager configuration, 90–91
 - consumer manager monitor, 91–95
 - defined, 19

MBeans (Continued)

- destination configuration, 71–75
- destination manager configuration, 81–85
- destination manager monitor, 85–87
- destination monitor, 75–81
- JVM monitor, 21, 111–112
- log configuration, 108–110
- log monitor, 110–111
- manager, defined, 22
- monitor, defined, 21
- notifications, receiving, 47–50
- operations, invoking, 39–47
- producer manager configuration, 87–88
- producer manager monitor, 88–90
- resource, defined, 21
- server
 - See* MBean server
- service configuration, 58–60
- service manager configuration, 63–64
- service manager monitor, 64–66
- service monitor, 60–63
- subtypes (table), 24
- transaction manager configuration, 95–96
- transaction manager monitor, 96–99
- types (table), 23–24
- using, 34–50

message consumers, 90–95

- acknowledgment mode, 93, 94, 95
- composite data object, lookup keys for (table), 93–94
- consumer identifier, 62, 68, 80, 91, 92, 93
- manager configuration MBean, 90–91
- manager monitor MBean, 91–95

message producers, 87–90

- composite data object, lookup keys for (table), 89–90
- manager configuration MBean, 87–88
- manager monitor MBean, 88–90
- producer identifier, 62, 68, 80, 88, 89

message string (notifications), 47

methods

- addNotificationListener (interface MBeanServerConnection), 47
- connect (class JMXConnectorFactory), 33

methods (Continued)

- createConnection (class AdminConnectionFactory), 32
- createConnectionConfig (class MQObjectName), 26, 66
- createConnectionMonitor (class MQObjectName), 26, 67
- createDestinationConfig (class MQObjectName), 26, 71
- createDestinationMonitor (class MQObjectName), 26, 76
- createServiceConfig (class MQObjectName), 26, 59
- createServiceMonitor (class MQObjectName), 26, 60
- enableType (class NotificationFilterSupport), 48
- getAttribute (interface MBeanServerConnection), 34
- getAttributes (interface MBeanServerConnection), 35
- getBrokerAddress (class BrokerNotification), 57
- getBrokerAddress (class ClusterNotification), 108
- getBrokerID (class BrokerNotification), 57
- getBrokerID (class ClusterNotification), 108
- getClusterID (class ClusterNotification), 108
- getConnectionID (class ConnectionNotification), 70
- getCreatedByAdmin (class DestinationNotification), 81, 86
- getDestinationName (class DestinationNotification), 81, 86
- getDestinationType (class DestinationNotification), 81, 86
- getFailedBrokerID (class BrokerNotification), 57
- getHeapMemoryUsage (class BrokerNotification), 58
- getLevel (class LogNotification), 111
- getMBeanServerConnection (class JMXConnector), 32
- getMessage (class LogNotification), 111
- getName (class Attribute), 35

methods (*Continued*)

- getNewResourceState (class BrokerNotification), 58
- getOldResourceState (class BrokerNotification), 58
- getPauseType (class DestinationNotification), 81,87
- getRemoteHost (class ConnectionNotification), 70
- getServiceName (class ConnectionNotification), 70
- getServiceName (class ServiceNotification), 63, 66
- getTransactionID (class TransactionNotification), 99
- getUserName (class ConnectionNotification), 70
- getValue (class Attribute), 35
- handleNotification (interface NotificationListener), 47
- invoke (interface MBeanServerConnection), 39,40
- isHighlyAvailable (class ClusterNotification), 108
- isMasterBroker (class ClusterNotification), 108
- setAttribute (interface MBeanServerConnection), 37
- setAttributes (interface MBeanServerConnection), 38
- setProperty (class AdminConnectionFactory), 32
- MinThreads attribute
 - service configuration MBean, 59
 - service manager configuration MBean, 64
- Monitor MBean subtype, 24
- monitor MBeans, 21
- mq.broker.quiesce.complete notification, broker monitor MBean, 57
- mq.broker.quiesce.start notification, broker monitor MBean, 57
- mq.broker.resource.state.change notification, broker monitor MBean, 57
- mq.broker.shutdown.start notification, broker monitor MBean, 57
- mq.broker.takeover.complete notification
 - broker monitor MBean, 57

- mq.broker.takeover.complete notification (*Continued*)
 - cluster monitor MBean, 108

- mq.broker.takeover.fail notification
 - broker monitor MBean, 57
 - cluster monitor MBean, 108
- mq.broker.takeover.start notification
 - broker monitor MBean, 57
 - cluster monitor MBean, 108
- mq.cluster.broker.down notification, cluster monitor MBean, 108
- mq.cluster.broker.join notification
 - broker monitor MBean, 57
 - cluster monitor MBean, 107
- mq.connection.close notification
 - connection manager monitor MBean, 70
 - service monitor MBean, 63
- mq.connection.open notification
 - connection manager monitor MBean, 70
 - service monitor MBean, 63
- mq.connection.reject notification
 - connection manager monitor MBean, 70
 - service monitor MBean, 63
- mq.destination.compact notification
 - destination manager monitor MBean, 86
 - destination monitor MBean, 81
- mq.destination.create notification, destination manager monitor MBean, 86
- mq.destination.destroy notification, destination manager monitor MBean, 86
- mq.destination.pause notification
 - destination manager monitor MBean, 86
 - destination monitor MBean, 81
- mq.destination.purge notification
 - destination manager monitor MBean, 86
 - destination monitor MBean, 81
- mq.destination.resume notification
 - destination manager monitor MBean, 86
 - destination monitor MBean, 81
- mq.log.level.ERROR notification, log monitor MBean, 110
- mq.log.level.INFO notification, log monitor MBean, 110

- mq.log.level.WARNING notification, log monitor MBean, 110
 - mq.service.pause notification
 - service manager monitor MBean, 65
 - service monitor MBean, 63
 - mq.service.resume notification
 - service manager monitor MBean, 65
 - service monitor MBean, 63
 - mq.transaction.commit notification, transaction manager monitor MBean, 99
 - mq.transaction.prepare notification, transaction manager monitor MBean, 99
 - mq.transaction.rollback notification, transaction manager monitor MBean, 99
 - MQNotification class, 30
 - MQObjectName utility class, 20, 25, 30
 - BROKER_CONFIG_MBEAN_NAME constant, 26, 52
 - BROKER_MONITOR_MBEAN_NAME constant, 26, 55
 - CLUSTER_CONFIG_MBEAN_NAME constant, 26, 100
 - CLUSTER_MONITOR_MBEAN_NAME constant, 26, 104
 - CONNECTION_MANAGER_CONFIG_MBEAN_NAME constant, 26, 68
 - CONNECTION_MANAGER_MONITOR_MBEAN_NAME constant, 26, 69
 - CONSUMER_MANAGER_CONFIG_MBEAN_NAME constant, 26, 91
 - CONSUMER_MANAGER_MONITOR_MBEAN_NAME constant, 26, 92
 - createConnectionConfig method, 26, 66
 - createConnectionMonitor method, 26, 67
 - createDestinationConfig method, 26, 71
 - createDestinationMonitor method, 26, 76
 - createServiceConfig method, 26, 59
 - createServiceMonitor method, 26, 60
 - DESTINATION_MANAGER_CONFIG_MBEAN_NAME constant, 26, 82
 - DESTINATION_MANAGER_MONITOR_MBEAN_NAME constant, 26, 85
 - JVM_MONITOR_MBEAN_NAME constant, 27, 111
 - LOG_CONFIG_MBEAN_NAME constant, 27, 109
 - LOG_MONITOR_MBEAN_NAME constant, 27, 110
 - PRODUCER_MANAGER_CONFIG_MBEAN_NAME constant, 26, 87
 - MQObjectName utility class (*Continued*)
 - PRODUCER_MANAGER_MONITOR_MBEAN_NAME constant, 26, 88
 - SERVICE_MANAGER_CONFIG_MBEAN_NAME constant, 26, 63
 - SERVICE_MANAGER_MONITOR_MBEAN_NAME constant, 26, 64
 - TRANSACTION_MANAGER_CONFIG_MBEAN_NAME constant, 26, 95
 - TRANSACTION_MANAGER_MONITOR_MBEAN_NAME constant, 26, 96
 - MsgBytesIn attribute
 - destination monitor MBean, 54, 78
 - service manager monitor MBean, 54, 65
 - service monitor MBean, 54, 61
 - MsgBytesOut attribute
 - destination monitor MBean, 54, 78
 - service manager monitor MBean, 54, 65
 - service monitor MBean, 54, 61
- ## N
- Name attribute
 - destination configuration MBean, 71, 83
 - destination monitor MBean, 76
 - service configuration MBean, 59
 - service monitor MBean, 61
 - name property (object name), 23
 - values (table), 25
 - NextMessageID attribute, destination monitor MBean, 78
 - NextMessageID lookup key, message consumer, 94
 - NO_ACKNOWLEDGE utility constant, interface Session, 95
 - NONE utility constant, class LogLevel, 109
 - Notification class, 47
 - notification filters, 47
 - notification listeners
 - defined, 47
 - example, 48
 - for log notifications, 110
 - registering, 49–50
 - notification objects
 - for broker notifications, 57
 - for cluster notifications, 108

- notification objects (*Continued*)
 - for connection notifications, 70
 - for connection service notifications, 66
 - defined, 47
 - for destination notifications, 81, 86
 - for log notifications, 110
 - for service notifications, 63
 - for transaction notifications, 99
- NotificationFilterSupport class, 48
- NotificationListener interface, 47
- notifications, MBean
 - alphabetical list (table), 122–123
 - broker configuration MBean, 55
 - broker monitor MBean, 56–58
 - cluster configuration MBean, 103
 - cluster monitor MBean, 107–108
 - connection manager monitor MBean, 70
 - defined, 20
 - destination configuration MBean, 75
 - destination manager configuration MBean, 84–85
 - destination manager monitor MBean, 86–87
 - destination monitor MBean, 81
 - log configuration MBean, 110
 - log monitor MBean, 110–111
 - receiving, 47–50
 - service configuration MBean, 60
 - service manager monitor MBean, 65–66
 - service monitor MBean, 62–63
 - transaction manager monitor MBean, 99
- NumAcks lookup key, transaction, 98
- NumActiveConsumers attribute, destination monitor MBean, 77
- NumActiveThreads attribute
 - service manager monitor MBean, 64
 - service monitor MBean, 61
- NumBackupConsumers attribute, destination monitor MBean, 77
- NumConnections attribute
 - connection manager configuration MBean, 68
 - connection manager monitor MBean, 69
 - service monitor MBean, 61
- NumConnectionsOpened attribute
 - connection manager monitor MBean, 54, 69
 - service monitor MBean, 54, 61
- NumConnectionsRejected attribute
 - connection manager monitor MBean, 54, 69
 - service monitor MBean, 54, 61
- NumConsumers attribute
 - connection monitor MBean, 68
 - consumer manager configuration MBean, 91
 - consumer manager monitor MBean, 92
 - destination monitor MBean, 76
 - service monitor MBean, 61
- NumDestinations attribute
 - destination manager configuration MBean, 82
 - destination manager monitor MBean, 85
- NumMsgs attribute
 - destination manager monitor MBean, 85
 - destination monitor MBean, 78
- NumMsgs lookup key
 - broker cluster, 106
 - message consumer, 94
 - message producer, 90
 - transaction, 98
- NumMsgsHeldInTransaction attribute, destination monitor MBean, 78
- NumMsgsIn attribute
 - destination monitor MBean, 54, 78
 - service manager monitor MBean, 54, 65
 - service monitor MBean, 54, 61
- NumMsgsInDMQ attribute, destination manager monitor MBean, 85
- NumMsgsOut attribute
 - destination monitor MBean, 54, 78
 - service manager monitor MBean, 54, 65
 - service monitor MBean, 54, 61
- NumMsgsPending lookup key, message consumer, 94
- NumMsgsPendingAcks attribute, destination monitor MBean, 78
- NumMsgsPendingAcks lookup key, message consumer, 94
- NumMsgsRemote attribute, destination monitor MBean, 78
- NumPktsIn attribute
 - service manager monitor MBean, 54, 65
 - service monitor MBean, 54, 61
- NumPktsOut attribute
 - service manager monitor MBean, 54, 65

NumPktsOut attribute (*Continued*)
 service monitor MBean, 54, 61

NumProducers attribute
 connection monitor MBean, 67
 destination monitor MBean, 76
 producer manager configuration MBean, 87
 producer manager monitor MBean, 88
 service monitor MBean, 61

NumServices attribute, service manager monitor MBean, 64

NumTransactions attribute
 transaction manager configuration MBean, 96
 transaction manager monitor MBean, 97

NumTransactionsCommitted attribute
 transaction manager monitor MBean, 55, 97

NumTransactionsRollback attribute
 transaction manager monitor MBean, 55, 97

NumWildcardConsumers attribute
 consumer manager monitor MBean, 92
 destination monitor MBean, 76

NumWildcardProducers attribute
 destination monitor MBean, 76
 producer manager configuration MBean, 88

NumWildcards attribute, destination monitor MBean, 77

O

object names, 23–27
 broker configuration MBean, 52
 broker monitor MBean, 55
 cluster configuration MBean, 100
 cluster monitor MBean, 104
 connection configuration MBean, 66
 connection manager configuration MBean, 68
 connection manager monitor MBean, 69
 connection monitor MBean, 67
 consumer manager configuration MBean, 91
 consumer manager monitor MBean, 92
 defined, 20
 destination configuration MBean, 71
 destination manager configuration MBean, 82
 destination manager monitor MBean, 85
 destination monitor MBean, 75–76

object names (*Continued*)
 desttype values (table), 24
 examples, 25
 JVM monitor MBean, 111
 log configuration MBean, 109
 log monitor MBean, 110
 name values (table), 25
 producer manager configuration MBean, 87
 producer manager monitor MBean, 88
 properties (table), 23
 service configuration MBean, 58–59
 service manager configuration MBean, 63
 service manager monitor MBean, 64
 service monitor MBean, 60–61
 subtype values (table), 24
 syntax, 23
 transaction manager configuration MBean, 95
 transaction manager monitor MBean, 96
 type values (table), 23–24
 utility constants and methods (table), 26–27

ObjectName class, 20, 23

OPERATING utility constant, class BrokerState, 107

operations, MBean
 alphabetical list (table), 119–121
 broker configuration MBean, 52–55
 cluster configuration MBean, 101–103
 cluster monitor MBean, 105–107
 combining with attributes, 42
 connection manager configuration MBean, 69
 connection manager monitor MBean, 70
 connection monitor MBean, 68
 consumer manager configuration MBean, 91
 consumer manager monitor MBean, 92–95
 defined, 20
 destination configuration MBean, 74–75
 destination manager configuration MBean, 83–84
 destination manager monitor MBean, 86
 destination monitor MBean, 79–81
 invoking, 39–47
 producer manager configuration MBean, 88
 producer manager monitor MBean, 88–90
 service configuration MBean, 60
 service manager configuration MBean, 64
 service manager monitor MBean, 65

operations, MBean (*Continued*)

- service monitor MBean, 62
- transaction manager configuration MBean, 96
- transaction manager monitor MBean, 97–99

P

packages

- com.sun.messaging, 29, 32
- com.sun.messaging.jms.management.server, 29

pause operation

- destination configuration MBean, 74, 75
- destination manager configuration MBean, 84
- service configuration MBean, 60
- service manager configuration MBean, 64

pause types, destination, 75

PAUSED utility constant

- class DestinationState, 79
- class ServiceState, 62

PeakMsgBytes attribute

- destination monitor MBean, 54, 78

PeakNumActiveConsumers attribute

- destination monitor MBean, 54, 77

PeakNumBackupConsumers attribute

- destination monitor MBean, 54, 77

PeakNumConsumers attribute

- destination monitor MBean, 54, 77

PeakNumMsgs attribute

- destination monitor MBean, 54, 78

PeakTotalMsgBytes attribute

- destination monitor MBean, 54, 78

PktBytesIn attribute

- service manager monitor MBean, 54, 65
- service monitor MBean, 54, 62

PktBytesOut attribute

- service manager monitor MBean, 54, 65
- service monitor MBean, 54, 62

Port attribute

- broker configuration MBean, 52
- broker monitor MBean, 56
- connection monitor MBean, 67
- service configuration MBean, 59
- service monitor MBean, 59, 61

PREPARED utility constant, class TransactionState, 98

PRODUCER_MANAGER_CONFIG_MBEAN_NAME utility

- constant
- class MQObjectName, 26, 87

producer manager configuration MBean, 87–88

- attribute, 87–88
- object name, 87
- operation, 88

producer manager monitor MBean, 88–90

- attribute, 88
- object name, 88
- operations, 88–90

PRODUCER_MANAGER_MONITOR_MBEAN_NAME utility

- constant
- class MQObjectName, 26, 88

ProducerAttributes utility class, 30, 87, 88

ProducerID lookup key, message producer, 89

ProducerInfo utility class, 31, 89

ProducerManager MBean type, 24

ProducerOperations utility class, 30, 88

producers, *See* message producers

PRODUCERS utility constant

- class DestinationPauseType, 75, 84

PRODUCERS_PAUSED utility constant, class

- DestinationState, 79

protocol types, 25

purge operation

- consumer manager configuration MBean, 91
- destination configuration MBean, 74

Q

q destination type, 24, 74, 79, 84, 90, 94

QUEUE utility constant

- class DestinationType, 24, 74, 79, 84, 90, 94

quiesce operation, broker configuration MBean, 53

QUIESCE_COMPLETE utility constant, class

- BrokerState, 107

QUIESCE_STARTED utility constant, class

- BrokerState, 107

QUIESCED utility constant, class ServiceState, 62

R

REJECT_NEWEST utility constant, class
 DestinationLimitBehavior, 74
 reload operation, cluster configuration MBean, 102
 ReloadXMLSchemaOnFailure attribute, destination
 configuration MBean, 73
 remote method invocation (RMI), 31
 REMOVE_LOW_PRIORITY utility constant, class
 DestinationLimitBehavior, 74
 REMOVE_OLDEST utility constant, class
 DestinationLimitBehavior, 74
 resetMetrics operation, broker configuration
 MBean, 53
 resource MBeans, 21
 ResourceState attribute, broker monitor MBean, 56
 restart operation, broker configuration MBean, 53
 resume operation
 destination configuration MBean, 74
 destination manager configuration MBean, 84
 service configuration MBean, 60
 service manager configuration MBean, 64
 RMI, *See* remote method invocation
 rollback operation, transaction manager configuration
 MBean, 96
 ROLLEDBACK utility constant, class
 TransactionState, 99
 RolloverBytes attribute, log configuration
 MBean, 109
 RolloverSecs attribute, log configuration MBean, 109
 RUNNING utility constant
 class DestinationState, 79
 class ServiceState, 62

S

Secure Hypertext Transfer Protocol (HTTPS), 25
 Secure Socket Layer (SSL), 25
 Selector lookup key, message consumer, 93
 server, MBean, *See* MBean server
 Service MBean type, 23
 service configuration MBean, 58–60
 attributes, 59–60
 notification, 60
 object name, 58–59

service configuration MBean (*Continued*)
 operations, 60
 SERVICE_MANAGER_CONFIG_MBEAN_NAME utility constant
 class MQObjectName, 26, 63
 service manager configuration MBean, 63–64
 attributes, 63–64
 object name, 63
 operations, 64
 service manager monitor MBean, 64–66
 attributes, 64–65
 notification objects, 66
 notifications, 65–66
 object name, 64
 operation, 65
 SERVICE_MANAGER_MONITOR_MBEAN_NAME utility
 constant
 class MQObjectName, 26, 64
 service monitor MBean, 60–63
 attributes, 61–62
 notification objects, 63
 notifications, 62–63
 object name, 60–61
 operations, 62
 SERVICE_PAUSE utility constant
 class ServiceNotification, 63, 65
 SERVICE_RESUME utility constant
 class ServiceNotification, 63, 65
 service URLs, JMX, parameter to
 JMXConnectorFactory.connect method, 34
 ServiceAttributes utility class, 30, 59, 61, 63, 64
 ServiceManager MBean type, 24
 ServiceName attribute, connection monitor
 MBean, 67
 ServiceName lookup key
 message consumer, 93
 message producer, 89
 ServiceNotification class, 30
 data retrieval method, 63, 66
 SERVICE_PAUSE constant, 63, 65
 SERVICE_RESUME constant, 63, 65
 utility constants, 62, 65
 ServiceOperations utility class, 30, 60, 62, 64, 65
 services, *See* connection services
 ServiceState utility class, 30, 62

ServiceState utility class (*Continued*)

- PAUSED constant, 62
- QUIESCED constant, 62
- RUNNING constant, 62
- UNKNOWN constant, 62

Session interface, 95

- AUTO_ACKNOWLEDGE constant, 95
- CLIENT_ACKNOWLEDGE constant, 95
- DUPS_OK_ACKNOWLEDGE constant, 95
- NO_ACKNOWLEDGE constant, 95
- SESSION_TRANSACTED constant, 95

SESSION_TRANSACTED utility constant, interface
Session, 95**setAttribute method, interface**

- MBeanServerConnection, 37

setAttributes method, interface

- MBeanServerConnection, 38

setProperty method, class

- AdminConnectionFactory, 32

shutdown operation

- broker configuration MBean, 53

SHUTDOWN_STARTED utility constant, class
BrokerState, 107**SSL, *See* Secure Socket Layer****ssladmin connection service name, 25****ssljms connection service name, 25****STARTED utility constant, class TransactionState, 98****State attribute**

- destination monitor MBean, 76, 79
- service monitor MBean, 61, 62

State lookup key

- broker cluster, 106, 107
- transaction, 98

StateLabel attribute

- destination monitor MBean, 76, 79
- service monitor MBean, 61, 62

StateLabel lookup key

- broker cluster, 106, 107
- transaction, 98

StatusTimestamp lookup key, broker cluster, 107**subtype property (object name), 23**
values (table), 24**T****t destination type, 24, 74, 79, 84, 90, 94****takeover operation, broker configuration MBean, 53****TAKEOVER_COMPLETE utility constant, class**
BrokerState, 107**TAKEOVER_FAILED utility constant, class**
BrokerState, 107**TAKEOVER_STARTED utility constant, class**
BrokerState, 107**TakeoverBrokerID lookup key, broker cluster, 106****TCP, *See* Transmission Control Protocol****Temporary attribute, destination monitor MBean, 76****ThreadPoolModel attribute, service configuration**
MBean, 59**TIMED_OUT utility constant, class**
TransactionState, 99**TLS, *See* Transport Layer Security****TOPIC utility constant**

- class DestinationType, 24, 74, 79, 84, 90, 94

TotalMemory attribute, JVM monitor MBean, 111**TotalMsgBytes attribute**

- destination manager monitor MBean, 85
- destination monitor MBean, 78

TotalMsgBytesHeldInTransaction attribute,
destination monitor MBean, 78**TotalMsgBytesInDMQ attribute, destination manager**
monitor MBean, 85**TotalMsgBytesRemote attribute, destination monitor**
MBean, 78**TRANSACTION_COMMIT utility constant, class**
TransactionNotification, 99**TRANSACTION_MANAGER_CONFIG_MBEAN_NAME utility**
constant
class MQObjectName, 26, 95**transaction manager configuration MBean, 95–96**
attribute, 96

- object name, 95
- operations, 96

transaction manager monitor MBean, 96–99
attributes, 97

- notification objects, 99
- notifications, 99
- object name, 96
- operations, 97–99

TRANSACTION_MANAGER_MONITOR_MBEAN_NAME utility constant
 class MQObjectName, 26, 96

TRANSACTION_PREPARE utility constant, class
 TransactionNotification, 99

TRANSACTION_ROLLBACK utility constant, class
 TransactionNotification, 99

TransactionAttributes utility class, 31, 96, 97

TransactionID lookup key, transaction, 98

TransactionInfo utility class, 31, 97

TransactionManager MBean type, 24

TransactionNotification class, 31
 data retrieval method, 99
 TRANSACTION_COMMIT constant, 99
 TRANSACTION_PREPARE constant, 99
 TRANSACTION_ROLLBACK constant, 99
 utility constants, 99

TransactionOperations utility class, 31, 96, 97

transactions, 95–99
 composite data object, lookup keys for (table), 98
 distributed transaction identifier, 98
 manager configuration MBean, 95–96
 manager monitor MBean, 96–99
 state values (table), 98–99
 transaction identifier, 96, 97, 98, 99

TransactionState utility class, 31, 98
 COMMITTED constant, 98
 COMPLETE constant, 98
 CREATED constant, 98
 FAILED constant, 98
 INCOMPLETE constant, 98
 PREPARED constant, 98
 ROLLEDBACK constant, 99
 STARTED constant, 98
 TIMED_OUT constant, 99
 UNKNOWN constant, 99

Transmission Control Protocol (TCP), 25

Transport Layer Security (TLS) protocol, 25

Type attribute
 destination configuration MBean, 71, 73, 83
 destination monitor MBean, 76, 79

type property (object name), 23
 values (table), 23–24

U

UNKNOWN utility constant
 class BrokerState, 107
 class DestinationState, 79
 class LogLevel, 109
 class ServiceState, 62
 class TransactionState, 99

getProperty operation, broker configuration
 MBean, 53

unquiesce operation, broker configuration MBean, 53

UseDMQ attribute
 destination configuration MBean, 72, 73

User attribute, connection monitor MBean, 67

user data object (notifications), 47

User lookup key
 message consumer, 93
 message producer, 89
 transaction, 98

UseSharedDatabaseForConfigRecord attribute
 cluster configuration MBean, 100
 cluster monitor MBean, 104

utility classes, 30–31
 AdminConnectionConfiguration, 29, 32
 BrokerAttributes, 30, 52, 55
 BrokerClusterInfo, 31, 102, 106
 BrokerOperations, 30, 52
 BrokerState, 30, 107
 ClusterAttributes, 31, 100, 104
 ClusterOperations, 31, 101, 105
 ConnectionAttributes, 30, 66, 67, 68, 69
 ConnectionOperations, 30, 68, 69, 70
 ConsumerAttributes, 31, 91, 92
 ConsumerInfo, 31, 93
 ConsumerOperations, 31, 91, 92
 DestinationAttributes, 30, 71, 76, 82, 83, 85
 DestinationLimitBehavior, 30, 74
 DestinationOperations, 30, 74, 79, 83, 86
 DestinationPauseType, 30, 75, 84
 DestinationState, 30, 79
 DestinationType, 24, 30, 73, 79, 84, 90, 94
 JVMAttributes, 31, 111
 LogAttributes, 31, 109
 LogLevel, 31, 109
 MQObjectName, 20, 25, 30

utility classes (*Continued*)

- ProducerAttributes, 30, 87, 88
- ProducerInfo, 31, 89
- ProducerOperations, 30, 88
- ServiceAttributes, 30, 59, 61, 63, 64
- ServiceOperations, 30, 60, 62, 64, 65
- ServiceState, 30, 62
- TransactionAttributes, 31, 96, 97
- TransactionInfo, 31, 97
- TransactionOperations, 31, 96, 97
- TransactionState, 31, 98

utility constants

- ALL (class DestinationPauseType), 75, 84
- AUTO_ACKNOWLEDGE (interface Session), 95
- BROKER_CONFIG_MBEAN_NAME (class MQObjectName), 26, 52
- BROKER_DOWN (class BrokerState), 107
- BROKER_MONITOR_MBEAN_NAME (class MQObjectName), 26, 55
- BROKER QUIESCE_COMPLETE (class BrokerNotification), 57
- BROKER QUIESCE_START (class BrokerNotification), 57
- BROKER_SHUTDOWN_START (class BrokerNotification), 57
- BROKER TAKEOVER_COMPLETE (class BrokerNotification), 57, 108
- BROKER_STATE_CHANGE (class BrokerNotification), 57
- BROKER TAKEOVER_FAIL (class BrokerNotification), 57, 108
- BROKER TAKEOVER_START (class BrokerNotification), 57, 108
- CLIENT_ACKNOWLEDGE (interface Session), 95
- CLUSTER_BROKER_DOWN (class ClusterNotification), 108
- CLUSTER_BROKER_JOIN (class ClusterNotification), 57, 107
- CLUSTER_CONFIG_MBEAN_NAME (class MQObjectName), 26, 100
- CLUSTER_MONITOR_MBEAN_NAME (class MQObjectName), 26, 104
- COMMITTED (class TransactionState), 98
- COMPLETE (class TransactionState), 98

utility constants (*Continued*)

- CONNECTION_CLOSE (class ConnectionNotification), 63, 70
- CONNECTION_MANAGER_CONFIG_MBEAN_NAME (class MQObjectName), 26, 68
- CONNECTION_MANAGER_MONITOR_MBEAN_NAME (class MQObjectName), 26, 69
- CONNECTION_OPEN (class ConnectionNotification), 63, 70
- CONNECTION_REJECT (class ConnectionNotification), 63, 70
- CONSUMER_MANAGER_CONFIG_MBEAN_NAME (class MQObjectName), 26, 91
- CONSUMER_MANAGER_MONITOR_MBEAN_NAME (class MQObjectName), 26, 92
- CONSUMERS (class DestinationPauseType), 75, 84
- CONSUMERS_PAUSED (class DestinationState), 79
- CREATED (class TransactionState), 98
- DESTINATION_COMPACT (class DestinationNotification), 81, 86
- DESTINATION_CREATE (class DestinationNotification), 86
- DESTINATION_DESTROY (class DestinationNotification), 86
- DESTINATION_MANAGER_CONFIG_MBEAN_NAME (class MQObjectName), 26, 82
- DESTINATION_MANAGER_MONITOR_MBEAN_NAME (class MQObjectName), 26, 85
- DESTINATION_PAUSE (class DestinationNotification), 81, 86
- DESTINATION_PURGE (class DestinationNotification), 81, 86
- DESTINATION_RESUME (class DestinationNotification), 81, 86
- DUPS_OK_ACKNOWLEDGE (interface Session), 95
- ERROR (class LogLevel), 109
- FAILED (class TransactionState), 98
- FLOW_CONTROL (class DestinationLimitBehavior), 74
- INCOMPLETE (class TransactionState), 98
- INFO (class LogLevel), 109
- JVM_MONITOR_MBEAN_NAME (class MQObjectName), 27, 111
- LOG_CONFIG_MBEAN_NAME (class MQObjectName), 27, 109

utility constants (*Continued*)

LOG_LEVEL_ERROR (class LogNotification), 110
 LOG_LEVEL_INFO (class LogNotification), 110
 LOG_LEVEL_WARNING (class LogNotification), 110
 LOG_MONITOR_MBEAN_NAME (class MQObjectName), 27, 110
 NO_ACKNOWLEDGE (interface Session), 95
 NONE (class LogLevel), 109
 OPERATING (class BrokerState), 107
 PAUSED (class DestinationState), 79
 PAUSED (class ServiceState), 62
 PREPARED (class TransactionState), 98
 PRODUCER_MANAGER_CONFIG_MBEAN_NAME (class MQObjectName), 26, 87
 PRODUCER_MANAGER_MONITOR_MBEAN_NAME (class MQObjectName), 26, 88
 PRODUCERS (class DestinationPauseType), 75, 84
 PRODUCERS_PAUSED (class DestinationState), 79
 QUEUE (class DestinationType), 24, 74, 79, 84, 90, 94
 QUIESCE_COMPLETE (class BrokerState), 107
 QUIESCE_STARTED (class BrokerState), 107
 QUIESCED (class ServiceState), 62
 REJECT_NEWEST (class DestinationLimitBehavior), 74
 REMOVE_LOW_PRIORITY (class DestinationLimitBehavior), 74
 REMOVE_OLDEST (class DestinationLimitBehavior), 74
 ROLLEDBACK (class TransactionState), 99
 RUNNING (class DestinationState), 79
 RUNNING (class ServiceState), 62
 SERVICE_MANAGER_CONFIG_MBEAN_NAME (class MQObjectName), 26, 63
 SERVICE_MANAGER_MONITOR_MBEAN_NAME (class MQObjectName), 26, 64
 SERVICE_PAUSE (class ServiceNotification), 63, 65
 SERVICE_RESUME (class ServiceNotification), 63, 65
 SESSION_TRANSACTED (interface Session), 95
 SHUTDOWN_STARTED (class BrokerState), 107
 STARTED (class TransactionState), 98
 TAKEOVER_COMPLETE (class BrokerState), 107

utility constants (*Continued*)

TAKEOVER_FAILED (class BrokerState), 107
 TAKEOVER_STARTED (class BrokerState), 107
 TIMED_OUT (class TransactionState), 99
 TOPIC (class DestinationType), 24, 74, 79, 84, 90, 94
 TRANSACTION_COMMIT (class TransactionNotification), 99
 TRANSACTION_MANAGER_CONFIG_MBEAN_NAME (class MQObjectName), 26, 95
 TRANSACTION_MANAGER_MONITOR_MBEAN_NAME (class MQObjectName), 26, 96
 TRANSACTION_PREPARE (class TransactionNotification), 99
 TRANSACTION_ROLLBACK (class TransactionNotification), 99
 UNKNOWN (class BrokerState), 107
 UNKNOWN (class DestinationState), 79
 UNKNOWN (class LogLevel), 109
 UNKNOWN (class ServiceState), 62
 UNKNOWN (class TransactionState), 99
 WARNING (class LogLevel), 109

utility methods

createConnectionConfig (class MQObjectName), 26, 66
 createConnectionMonitor (class MQObjectName), 26, 67
 createDestinationConfig (class MQObjectName), 26, 71
 createDestinationMonitor (class MQObjectName), 26, 76
 createServiceConfig (class MQObjectName), 26, 59
 createServiceMonitor (class MQObjectName), 26, 60

V

ValidateXMLSchemaEnabled attribute, destination configuration MBean, 73
 Version attribute
 broker configuration MBean, 52
 broker monitor MBean, 56

W

WARNING utility constant, class `LogLevel`, 109

Wildcard lookup key

- message consumer, 93

- message producer, 90

X

XID, *See* distributed transaction identifier

XID lookup key, transaction, 98

XMLSchemaURLList attribute, destination configuration

- MBean, 73