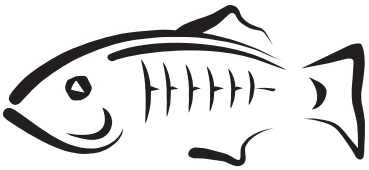


Open Message Queue

Release Notes

Release 5.1.1

September 2017

This book describes new features, compatibility issues, and existing bugs for the Message Queue 5.1.1, 5.1 and earlier releases.

Open Message Queue Release Notes, Release 5.1.1

Preface

This book provides information about concepts and procedures for developing Java messaging applications (Java clients) that work with Message Queue.

This preface consists of the following sections:

Documentation Conventions

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

Typographic Conventions

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

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

Symbol Conventions

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

Symbol	Description	Example	Meaning
`[ ]`	Contains optional arguments and command options.	`ls [-l]`	The `-l` option is not required.
`{ | }`	Contains a set of choices for a required command option.	`-d {y|n}`	The `-d` option requires that you use either the `y` argument or the `n` argument.
`${ }`	Indicates a variable reference.	`${com.sun.javaRoot}`	References the value of the `com.sun.javaRoot` 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.

Shell	Prompt
C shell on UNIX, Linux, or AIX	*machine-name*`%`
C shell superuser on UNIX, Linux, or AIX	*machine-name*`#`
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.

Variable	Description
IMQ_HOME	The Message Queue home directory: ■ For installations of Message Queue bundled with GlassFish Server, IMQ_HOME is *as-install-parent*/mq, where *as-install-parent* is the parent directory of the GlassFish Server base installation directory, glassfish5 by default for MQ 5.1.1 and glassfish4.1 for MQ 5.0. ■ For installations of Open Message Queue, IMQ_HOME is *mqInstallHome*/mq.
IMQ_VARHOME	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: ■ For installations of Message Queue bundled with GlassFish Server, IMQ_VARHOME defaults to *as-install-parent*/glassfish/domains/domain1/imq. ■ For installations of Open Message Queue, IMQ_HOME defaults to *mqInstallHome*/var/mq.
IMQ_JAVAHOME	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.

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
- Java Message Service (JMS) Specification
- JavaDoc
- Example Client Applications
- Online Help

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 at https://javaee.github.io/openmq/Documentation.html.

Document	Audience	Description
Technical Overview	Developers and administrators	Describes Message Queue concepts, features, and components.
Release Notes	Developers and administrators	Includes descriptions of new features, limitations, and known bugs, as well as technical notes.
Administration Guide	Administrators, also recommended for developers	Provides background and information needed to perform administration tasks using Message Queue administration tools.
Developer's Guide for Java Clients	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.

Document	Audience	Description
Developer's Guide for C Clients	Developers	Provides programming and reference documentation for developers of C client programs using the Message Queue C implementation of the JMS API (C-API).
Developer's Guide for JMX Clients	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 https://javaee.github.io/jms-spec/.

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.

For 5.1 and 5.1.1: Example C client applications are available in MQ source code bundles at https://github.com/javaee/openmq/releases

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 "Command Line Reference" in *Open Message Queue Administration Guide*. The Message Queue graphical user interface (GUI) administration tool, the Administration Console, also includes a context-sensitive help facility; for details, see "Administration Console Online Help" in *Open Message Queue Administration Guide*.

Documentation

For additional information see:

- Documentation

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at
http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit
http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit
http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

1

Release Notes

These release notes contain important information available at the time of release of Message Queue 5.1.1. New features and enhancements, known issues and limitations, and other information are addressed here. Read this document before you begin using Message Queue 5.1.1.

The most up-to-date version of these release notes can be found at `https://javaee.github.io/openmq/Documentation.html`

Check the web site prior to installing and setting up your software and then periodically thereafter to view the most up-to-date release notes and product.

These release notes contain the following sections:

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

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

Release Notes Revision History

The following table lists the dates for all 4.x and 5.x releases of the Message Queue product and describes the changes in this document associated with each release.

Table 1–1 Revision History

Date	Description of Changes
September 2017	Release of this document for Message Queue 5.1.1
September 2014	Release of this document for Message Queue 5.1
June 2013	Release of this document for Message Queue 5.0.
May 2013	Release of this document for Message Queue 4.5.2.1.
February 2012	Release of this document for Message Queue 4.5.2.
July 2011	Second release of this document for Message Queue 4.5. Corrects errors and omissions, and adds information about bug 6804819.
February 2011	Release of this document for Message Queue 4.5.
June 2010	Release of this document for Message Queue 4.4.2.
June 2010	Second release of this document for Message Queue 4.4 Update 1. Corrects errors and omissions, and adds information about bug 6925362.
December 2009	Release of this document for Message Queue 4.4 Update 1. Adds new features for this release and removes outdated installation issues that applied to the previous installation program.
December 2009	Second release of this document for Message Queue 4.4. Corrects errors and omissions.
October 2009	Release of this document for Message Queue 4.4. Adds new features for this release.
May 2009	Initial release of this document for Message Queue 4.4 Beta. Adds new features for this release.
December 2008	Release of this document for Message Queue 4.3. Adds new features for this release.
August 2008	Release of this document for Message Queue 4.2. Adds new features for this release.
September 2007	Third release of this document for Message Queue 4.1. Adds description of support for Java Enterprise System Monitoring Framework, fixed C ports, bug fixes, and other features.
April 2007	Second release of this document for Message Queue 4.1 Beta. Adds high availability feature.
January 2007	Initial release of this document for Message Queue 4.1 Beta. Adds description of JAAS support.
May 2006	Initial release of this document for Message Queue 4.0.

About Message Queue 5.1.1

Message Queue is a full-featured message service that provides reliable, asynchronous messaging in conformance with the Java Messaging Specification (JMS). Message Queue 5.1.1 conforms to JMS 2.0.1 and is integrated into GlassFish 5.0 which conforms to the Java Platform, Enterprise Edition 8 Specification (Java EE 8). In addition, Message Queue provides features that go beyond the JMS specification to meet the needs of large-scale enterprise deployments.

Message Queue 5.1.1 is an incremental release participating in the 5.0 release of GlassFish Server. Message Queue 5.1.1 can also be downloaded from https://javaee.github.io/openmq/Downloads.html

Unlike previous releases of Message Queue, the Message Queue 5.1.1 binary release does not include the C-API library although the C-API source code is available at https://github.com/javaee/openmq

Bugs Fixed in Message Queue 5.1.1

The following table lists the bugs fixed in Message Queue 5.1.1. Some of these issues are marked with "(OpenMQ)", which indicates the issue was reported in the issue tracker of the Open Message Queue open source project upon which Oracle GlassFish Server Message Queue is based.

The following table lists the bugs fixed in Message Queue 5.1.1

Table 1–2 Bugs Fixed in Message Queue 5.1.1

Bug	Description
20402088	Broker HA monitor thread should be daemon thread
19906529	Cluster listener thread exit when rogue client send pkt with huge pkt size field
12296963	Fix for Sybase "INCORRECT SYNTAX NEAR THE KEYWORD 'UNION'

Note: There are some additional issues listed at github location OpenMQ Issues that are addressed in Message Queue 5.1.1.

About Message Queue 5.1

Message Queue is a full-featured message service that provides reliable, asynchronous messaging in conformance with the Java Messaging Specification (JMS) 2.0 and the Java Platform, Enterprise Edition 7 Specification (Java EE 7). In addition, Message Queue provides features that go beyond the JMS specification to meet the needs of large-scale enterprise deployments.

Message Queue 5.1 is an incremental release participating in the 4.1 release of GlassFish Server. As a consequence, no separately downloadable, installable distribution of Message Queue 5.1 is available.

Message Queue 5.1 Supported Platforms and Components

This section covers the following topics regarding Message Queue 5.1 system requirements:

- Platform Support
- System Virtualization Support
- Optional Support Components

Platform Support

As a participant in the 4.1 release of GlassFish Server, Message Queue 5.1 supports the operating environments, databases, LDAP servers, and hardware listed in the *Oracle*

GlassFish Server 4.1 Certification Matrix, which is accessible at (https://javaee.github.io/glassfish/).

System Virtualization Support

System virtualization is a technology that enables multiple operating system (OS) instances to execute independently on shared hardware. Functionally, software deployed to an OS hosted in a virtualized environment is generally unaware that the underlying platform has been virtualized. Oracle performs testing of its products on select system virtualization and OS combinations to help validate that Oracle products continue to function on properly sized and configured virtualized environments as they do on non-virtualized systems.

Optional Support Components

In addition to the software components listed in the *Oracle GlassFish Server 4.1 Certification Matrix*, Table 1–3 shows components that you can install to provide additional support for Message Queue clients.

Table 1–3 Optional Support Components

Component	Supports	Supported Versions
Java Naming and Directory Interface (JNDI)	Administered object support and LDAP user repository	JNDI Version 1.2.1 LDAP Service Provider, Version 1.2.2
C Compiler and compatible C++ runtime library	Message Queue C clients	**Solaris:** Oracle Solaris Studio, Version 12 or later, C++ compiler with standard mode and C compiler **Linux:** gcc/g++, Version 3.4.6 **Windows:** Microsoft Windows Visual Studio, Version 2008 SP1
Netscape Portable Runtime (NSPR)	Message Queue C clients	Version 4.8.6
Network Security Services (NSS)	Message Queue C clients	Version 3.12.8

New Features in Message Queue 5.1

Message Queue 5.1 provides support for the Java EE 7 release. It includes new features, some feature enhancements, and bug fixes. This section includes a description of new features in this releases:

MQ JMS Client over WebSocket

MQ has traditionally supported HTTP Servlet Tunneling for MQ Java clients to communicate with a message broker over HTTP/HTTPS transport protocol. This new feature allows MQ JMS clients to communicate with MQ broker over WebSocket transport. Please see details at https://javaee.github.io/openmq/www/5.0.1/ws.html

MQ STOMP Client over WebSocket

STOMP is a simple text streaming oriented messaging protocol which provides interoperable wire format for any STOMP client to communicate with a STOMP messaging broker. MQ broker has provided STOMP messaging service via the 'stomp' bridge, which supports STOMP on TCP or SSL transport. This new feature allows

STOMP clients communicate to MQ broker over WebSocket. Please see details at https://javaee.github.io/openmq/www/5.0.1/ws.html

Bugs Fixed in Message Queue 5.1

The following table lists the bugs fixed in Message Queue 5.1. Some of these issues are marked with "(OpenMQ)", which indicates the issue was reported in the issue tracker of the Open Message Queue open source project upon which Oracle GlassFish Server Message Queue is based.

The following table lists the bugs fixed in Message Queue 5.1

Table 1–4 Bugs Fixed in Message Queue 5.1

Bug	Description
18918671	A broker thread removing temp destination can deadlock with temp destination's reconnect reaper thread
18868362	imqbrokerd -startrmiregistry -usermiregistry option precedence order incorrect
18434462	Persisting in message store within synchronized code is extremely non-scalable
18125457	Remove IMQVARHOME/IMQHOME information from portmapper output
17738518	Session.commit should auto-rollback the transaction if broker returns Status.GONE
17317188	'imqcmd restart broker' should always pass "nofailover=true' to broker
17316839	accesscontrol: produce.allow '*' and produce.deny combination not work as expected
17313998	JDBC connection pool reaper thread logs NPE if no idle connection.

Note: There are some additional issues listed at github location OpenMQ Issues that are addressed in Message Queue 5.1.

Installation

Message Queue 5.1 is installed as a sub-directory of the GlassFish 4.1 installation. For installation information, see the *GlassFish Server Open Source Edition Installation Guide*.

Compatibility Considerations

This section covers compatibility considerations when using Message Queue.

- Message Queue 5.1 must be used with Java SE 7. This general JMS 2.0 and Java EE 7 requirement implies that whenever Message Queue 5.1 jars are used in your classpath, you must use Java 7. For information on how to set the Java runtime for a broker, see "Using an "Alternative Java Runtime" in the *Open Message Queue Administration Guide*.
- Message Queue 5.1 brokers now use the Java `java.util.logging` logger.
- Message Queue uses many interfaces that may change over time. Scalability of Message Queue Interfaces in *Open Message Queue Administration Guide* classifies the interfaces according to their stability. The more stable an interface, the less likely it is to change in subsequent versions of the product.
- HADB database is no longer supported in the Message Queue 5.1 release.

Features to be Deprecated in a Future Release

The following features will be deprecated in a future release:

- **Message-based monitoring**

  Message-based monitoring makes use of the broker's configurable Metrics Message Producer to write metrics data into JMS messages, which are then sent to metrics topic destinations, depending on the type of metrics information contained in the messages. This metrics information can then be accessed by writing a client application that subscribes to the appropriate metrics topic destination, consumes its messages, and processes the data as desired.

  The message-based monitoring feature has been supplanted by the Administration API that was introduced in MQ 4.0 (see Support for JMX Administration API). The JMX API is more comprehensive (it includes more metrics data than is written to topic destinations) and is based on the JMX industry standard.

  There is no compelling reason to use message-based monitoring now that Message Queue supports the JMX API. Information about message-based monitoring will remain in the Message Queue JMX until the feature is formally deprecated.

- **Clear Text Passfile**

  Using a clear text passfile is not recommended and support will be removed in a future release. Oracle recommends existing plain text passfiles be obfuscated by running `imqusermgr encode`. See "Password Files" in the *Open Message Queue Administration Guide*.

Known Issues and Limitations

This section contains a list of the known issues with Message Queue 5.1. The following product areas are covered:

- Deprecated Password Option
- Administration/Configuration Issues
- Broker Issues
- Broker Clusters
- SOAP Support

For a list of current bugs, their status, and workarounds, see the OpenMQ Issues. Please check that page before you report a new bug. Although all Message Queue bugs are not listed, the page is a good starting place if you want to know whether a problem has been reported.

To report a new bug or submit a feature request, please file an issue at `https://github.com/javaee/openmq/`.

Deprecated Password Option

In previous versions of Message Queue, you could use the `–p` or `–password` option to specify a password interactively for the following commands: `imqcmd`, `imqbrokerd`, and `imdbmgr`. Beginning with version 4.0, these options have been deprecated.

Instead, you can create a password file that specifies the relevant passwords and reference the password file using the `-passfile` command option, or simply enter a password when prompted by the command.

A password file can contain one or more of the passwords listed below.

- A keystore password used to open the SSL keystore. Use the `imq.keystore.password` property to specify this password.
- An LDAP repository password used to connect securely with an LDAP directory if the connection is not anonymous. Use the `imq.user_repository.ldap.password` property to specify this password.
- A JDBC database password used to connect to a JDBC-compliant database. Use the `imq.persist.jdbc.vendorName.password` property to specify this password. The *vendorName* component of the property name is a variable that specifies the database vendor. Choices include `hadb`, `derby`, `pointbase`, `oracle`, or `mysql`.
- A password to the `imqcmd` command (to perform broker administration tasks). Use the `imq.imqcmd.password` property to specify this password.

In the following example, the password to the JDBC database is set in the password file to `abracadabra`.

```
imq.persist.jdbc.mysql.password=abracadabra
```

You can use a password file in one of the following ways.

- Configure the broker to use the password file by setting the following properties in the broker's `config.properties` file.

  `imq.passfile.enabled=trueimq.passfile.dirpath=`*passwordFileDirectory*`imq.passfile.name=`*passwordFileName*

- Use the `-passfile` option of the relevant command, for example:

  `imqbrokerd -passfile` *passwordFileName*

Administration/Configuration Issues

The following issues pertain to administration and configuration of Message Queue.

- On Windows platforms, you need to manually add the Message Queue broker as a Windows service using the `imqsvcadm` command. The installer does not do this for you.
- On Windows platforms, the built-in Windows Firewall, which is enabled by default, must be manually configured with a firewall rule that allows the broker to accept incoming connections from clients. (*Bug 6675595*)

  1. Double-click on Windows Firewall in the Control Panel

     You will have to click Continue on the User Account Control dialog for the Windows Firewall Settings dialog to open.

  2. In the Windows Firewall Settings dialog, click the Exceptions tab.
  3. Click Add program.
  4. In the Add a Program dialog, select `java.exe` and click Browse.

     Windows identifies the broker process as a Java Platform SE binary. Therefore, locate the `java.exe` used by the broker.

  5. Click Change scope.
  6. In the Change Scope dialog, select "Any computer (including those on the Internet."
  7. Click OK.

8. In the Add a Program dialog, click OK.
9. In the Windows Firewall Settings dialog, click OK.

- On Windows platforms, the `imqadmin` and `imqobjmgr` commands throw an error when the `CLASSPATH` contains double quotes. (*Bug 5060769*)

  Workaround: Open a command prompt window and unset the `CLASSPATH`:

  ```
  set classpath=
  ```

  Then run the desired command the same command prompt window, for example:

  mqInstallHome`\mq\bin\imqadmin`

- The `-javahome` option in all Solaris and Windows scripts does not work if the value provided contains a space. (*Bug 4683029*)

  The `javahome` option is used by Message Queue commands and utilities to specify an alternate Java compatible runtime to use. However, the path name to the alternate Java runtime must not contain spaces. The following are examples of paths that include spaces.

  Windows: `C:\jdk 1.7`

  Solaris: `/work/java 1.7`

  Workaround: Install the Java runtime at a location or path that does not contain spaces.

- The `imqQueueBrowserMaxMessagesPerRetrieve` attribute specifies the maximum number of messages that the client runtime retrieves at one time when browsing the contents of a queue. The attribute affects how the queued messages are batched, to be delivered to the client runtime, but it does not affect the total number of messages browsed. The attribute only affects the browsing mechanism, it does not affect queue message delivery. *(Bug 6387631)*

- On Linux platform running SELinux, the Update Center `pkg` command fails *(Bug 6892062)*

  Workaround: This issue is caused by a known issue in Update Center `UPDATECENTER2-1211` (. Use the following command to enable `pkg` to function on SELinux with enforcement enabled:

  ```
  # chcon -f -t textrel_shlib_t $IMAGE/pkg/vendor-packages/OpenSSL/crypto.so
  ```

Broker Issues

- When a JMS client using the HTTP connection service terminates abruptly (for example, using `Ctrl-C`) the broker takes approximately one minute before releasing the client connection and all the associated resources.

  If another instance of the client is started within the one minute period and if it tries to use the same ClientID, durable subscription, or queue, it might receive a "Client ID is already in use" exception. This is not a real problem; it is just the side effect of the termination process described above. If the client is started after a delay of approximately one minute, everything should work fine.

Broker Clusters

- A client can only browse the contents of queues that are located on its home broker. The client can still send messages to any queue or consume messages from any queue in the cluster; the limitation only affects queue browsing.
- In a conventional cluster that includes version 4.3 brokers, all brokers must be version 3.5 or later.
- When converting from a conventional cluster to an enhanced cluster, you can use the Message Queue Database Manager utility (`imqdbmgr`) to convert an existing standalone JDBC-based data store to a shared JDBC data store as documented in "Cluster Conversion: JDBC-Based Data Store" in *Open Message Queue Administration Guide*.

SOAP Support

You need to be aware of two issues related to SOAP support

- Beginning with the release of version 4.0 of Message Queue, support for SOAP administered objects is discontinued.
- SOAP development depends upon several files: `SUNWjaf`, `SUNWjmail`, `SUNWxsrt`, and `SUNWjaxp`. In version 4.1 of Message Queue, these files are available to you only if you are running Message Queue with JDK version 1.6.0 or later.
- Previously the SAAJ 1.2 implementation .jar directly referenced `mail.jar`. In SAAJ 1.3 this reference was removed; thus, Message Queue clients must explicitly put `mail.jar` in `CLASSPATH`.

Redistributable Files

Oracle GlassFish Server Message Queue contains the following set of files which you may use and freely distribute in binary form:

`fscontext.jar`
`imq.jar`
`imqjmx.jar`
`imqxm.jar`
`imqums.war`
`jaxm-api.jar`
`jms.jar`
`libmqcrt.sl` (HP-UX)
`libmqcrt.so` (UNIX)
`mqcrt1.dll` (Windows)
In addition, you can also redistribute the `LICENSE` and `COPYRIGHT` files.

Additional Resources

Useful Message Queue information can be found at the following Internet locations:

- Open Message Queue (Open MQ) website

  https://javaee.github.io/openmq/

- Java Message Service Specification website

  https://javaee.github.io/jms-spec/

New Features in Previous Message Queue 5.0

Message Queue 5.0 is a minor release providing support for the Java Messaging Specification (JMS), version 2.0 and the Java EE 7 release. It included a few new features, some feature enhancements, and bug fixes. This section includes a description of new features in this releases:

Support for JMS 2.0 Features and Enhancements

Message Queue 5.0 implements the JMS 2.0 API. This introduces a completely new Simplified API that makes JMS much simpler and easier to use. The existing Classic API remains and a number of improvements have been made to make the Classic API simpler and easier to use as well. For more information, see "The JMS Simplified API" in *Open Message Queue Developer's Guide for Java Clients*.

Other changes introduced into JMS 2.0 include:

- Designating a topic subscription as being *shared*, which allows it to have more than one consumer. Setting `clientId` is optional for shared subscriptions.
- A new method `getBody` has been added to `Message` which allows the message body to be extracted without the need to cast to a particular subtype.
- A new method, `setDeliveryDelay`, has been added to `MessageProducer` which allows a delivery delay to be specified. A message will not be delivered to a consumer until after the specified delay has elapsed.
- New send methods have been added to `MessageProducer` which allow messages to be sent asynchronously. These methods permit the JMS provider to perform part of the work involved in sending the message in a separate thread. When the send is complete, a callback method is invoked on an object supplied by the caller.
- The `Connection`, `Session`, `MessageProducer`, `MessageConsumer` and `QueueBrowser` interfaces have been modified to extend the `java.lang.Autocloseable` interface. This means that applications can create these objects using a Java SE 7 `try-with-resources` statement which removes the need for applications to explicitly call `close()` when these objects are no longer required.
- The existing standard message property `JMSXDeliveryCount` has been made mandatory. It was previously optional. This means that Message Queue will now always set this property to the number of times the message has been delivered.

Additional Message Queue 5.0 Enhancements

This release of Message Queue also includes the following changes and enhancements:

- Previously, the `JMXDeliveryCount` was used as a property to track the number of times a message was delivered to a given consumer before being placed on the DMQ. To conform to the JMS 2.0 specification, this Message Queue release introduces `JMS_SUN_DMQ_DELIVERY_COUNT` as a new property for that purpose.
- A new connection factory property, `imqAsyncSendCompletionWaitTimeout`, sets the amount of time, in milliseconds, that a MQ client waits for an asynchronous send to complete before calling `CompletionListener.onException`.
- The *shared* `threadpool_model` for a connection service that was used in previous releases has been replaced by a new implementation and the *shared* `threadpool_model` is now able to support `tls` protocoltype.

- A new administrative interface to provide the ability to obfuscate passwords in a `passfile` for Message Queue broker command line utilities. See "Password Files" in the *Open Message Queue Administration Guide*.
- Support for DB reconnect in the Message Queue JDBC Connection Pool. See "JDBC-Based Persistence" and "To Connect Brokers Using a Cluster Configuration File" in the *Open Message Queue Administration Guide*.
- The following C API functions are added this release to support shared durable subscribers:
  - `MQCreateSharedDurableMessageConsumer`
  - `MQCreateSharedMessageConsumer`
  - `MQCreateAsyncSharedDurableMessageConsumer`
  - `MQCreateAsyncSharedMessageConsumer`

  See "Reference" in the *Open Message Queue Developer's Guide for C Clients*.
- The following C API functions were added to support message delivery delay:
  - `MQGetDeliveryDelay` function
  - `MQSetDeliveryDelay` function
  - `MQ_DELIVERY_TIME_HEADER_PROPERTY` property

  See "Reference" in the *Open Message Queue Developer's Guide for C Clients*.
- The `NumMsgsInDelayDelivery` attribute was added to the DestinationMonitor MBean. See "Message Queue MBean Reference" in *Open Message Queue Developer's Guide for JMX Clients*.

New Features in Previous Message Queue 4 Releases

The new features in previous releases of the Message Queue 4 family are described in the following sections:

- New Features in Message Queue 4.5
- New Features in Message Queue 4.4.2
- New Features in Message Queue 4.4 Update 1
- New Features in Message Queue 4.4
- New Features in Message Queue 4.3
- New Features in Message Queue 4.2
- New Features in Message Queue 4.1
- New Features in Message Queue 4.0

New Features in Message Queue 4.5

Message Queue 4.5 is an incremental release that includes a number of feature enhancements and bug fixes. Two of the most important features in this release relate to broker clusters, and another relates to consumer event notifications for Java clients:

Conventional clusters of peer brokers

This release introduces a new type of conventional cluster, the conventional cluster of peer brokers. Unlike a conventional cluster with a master broker, a conventional cluster of peer brokers maintains the cluster configuration change record in a shared

JDBC data store instead of in the master broker. Thus, brokers can access cluster configuration information whether any other brokers in the cluster are running or not. For more information about conventional clusters of peer brokers, see "Broker Clusters" in *Open Message Queue Technical Overview*. For information about configuring and managing conventional clusters of peer brokers, see "Configuring and Managing Broker Clusters" in *Open Message Queue Administration Guide*.

Dynamically changing the master broker

Previously, to change the master broker in a conventional cluster from one broker to another, you had to stop all brokers, manually migrate the cluster configuration change record from the old master broker to the new one, and then start all brokers. This release provides the ability to change the master broker dynamically without stopping the cluster or performing manual migration tasks. For more information, see "Changing the Master Broker in a Conventional Cluster with Master Broker" in *Open Message Queue Administration Guide*.

Consumer event notifications for Java clients

This release introduces consumer event notifications for Java clients, which allow a Java client to listen for the existence of consumers on a destination. Thus, for example, a producer client can start or stop producing messages to a given destination based on the existence of consumers on the destination. For more information, see "Consumer Event Notification" in *Open Message Queue Developer's Guide for Java Clients*.

New Features in Message Queue 4.4.2

Message Queue 4.4.2 is a minor release that includes a number of feature enhancements and bug fixes. This section describes the new features included in this release.

- Message Queue now supports literal IPv6 addresses as broker host names when the *hostname:port* format is used. Previously, literal IPv6 addresses were only supported for the *hostname* format. If you use a literal IPv6 address, its format must conform to RFC2732 (`http://www.ietf.org/rfc/rfc2732.txt`), *Format for Literal IPv6 Addresses in URL's*.
- To address situations related to failover and restart of brokers in enhanced clusters, these features have been added:
  - The `-reset takeover-then-exit` option of the `imqbrokerd` command
  - The `imq.cluster.ha.takeoverWaitTimeout` broker property
- To provide more configurable control of connections to a JDBC data store, these broker properties have been added:
  - `imq.persist.jdbc.connection.timeoutIdle`
  - `imq.persist.jdbc.connection.validateOnGet`
  - `imq.persist.jdbc.connection.validationQuery`
- To control generation of informational log messages about successful message transfers across a JMS bridge, the `log-message-transfer` attribute has been added to the `jmsbridge` element in the XML configuration file for a JMS bridge.
- To enable the STOMP bridge service to bind to a specific network interface, the `imq.bridge.stomp.hostname` broker property has been added.

New Features in Message Queue 4.4 Update 1

Message Queue 4.4 Update 1 is a minor release that includes a number of feature enhancements and bug fixes. This section describes the new features included in this release:

New Installation Program

Message Queue 4.4 Update 1 provides a new multiplatform installer based on the `pkg(5)` system, also known as IPS or Image Packaging System. For information about this installer, see the *Sun GlassFish Message Queue 4.4 Update 1 Installation Guide*.

Transaction Log Support for Clusters

Message Queue 4.4 Update 1 adds a transaction persistence mechanism for file-based data stores that supports broker clusters. This mechanism provides other features as well, as described in "Optimizing File-Based Transaction Persistence" in *Open Message Queue Administration Guide*.

In-Process Broker

Message Queue 4.4 Update 1 supports running a broker from within a Java client. Such a broker, called an *in-process* or *embedded* broker, runs in the same JVM as the Java client that creates and starts it. For more information, see "Embedding a Message Queue Broker in a Java Client" in *Open Message Queue Developer's Guide for Java Clients*.

New Features in Message Queue 4.4

Message Queue 4.4 is a minor release that includes a number of feature enhancements and bug fixes. This section describes the new features included in this release:

JMS Bridge Service

Because the JMS specification does not define a wire protocol for communication between brokers and clients, each JMS provider (including Message Queue) has defined and uses its own propriety protocol. This situation has led to non-interoperability across JMS providers.

The JMS bridge service in Message Queue 4.4 closes this gap by enabling a Message Queue broker to map its destinations to destinations in external JMS providers. This mapping effectively allows the Message Queue broker to communicate with clients of the external JMS provider.

The JMS bridge service supports mapping destinations in external JMS providers that:

- Are JMS 1.1 compliant
- Support JNDI administrative objects
- Use connection factories of type `javax.jms.ConnectionFactory` or `javax.jms.XAConnectionFactory`

- For transacted mapping, support the XA interfaces as a resource manager

Many open source and commercial JMS providers meet these requirements, which makes the JMS bridge service an effective way to integrate Message Queue into an existing messaging environment that employs other JMS providers.

For more information about the JMS bridge service see "Configuring and Managing JMS Bridge Services" in *Open Message Queue Administration Guide*.

STOMP Bridge Service

As mentioned earlier, the JMS specification does not define a wire protocol for communication between brokers and clients. The STOMP (Streaming Text Oriented Messaging Protocol) open source project at `http://docs.codehaus.org/display/STOMP` defines a simple wire protocol that clients written in any language can use to communicate with any messaging provider that supports the STOMP protocol.

Message Queue 4.4 provides support for the STOMP protocol through the STOMP bridge service. This service enables a Message Queue broker communicate with STOMP clients.

For more information about the STOMP bridge service see "Configuring and Managing STOMP Bridge Services" in *Open Message Queue Administration Guide*.

Additional Enhancements

The following additional enhancements are also provided in Message Queue 4.4:

- New Universal Message Service (UMS) Functions
- IPS Package Support
- Audit Logging Feature Reinstated

New Universal Message Service (UMS) Functions The UMS now provides functions that use HTTP GET to offer several services:

- **getBrokerInfo**: retrieves information about the broker.
- **getConfiguration**: retrieves information about the UMS configuration.
- **debug**: turns debug logging in the UMS server on and off.
- **ping**: communicates with the broker to confirm that it is running.

For information about these new features, see "Query and utility functions using HTTP GET" in `http://mq.java.net/4.4-content/imqums/protocol.html`.

For an overview of UMS, see Universal Message Service (UMS). For of the UMS API, see `http://mq.java.net/4.4-content/imqums/protocol.html`. For programming examples in several languages, see `http://mq.java.net/4.4-content/imqums/examples/README.html`.

IPS Package Support Message Queue is now packaged for distribution using the open source Image Packaging System (IPS), also known as the `pkg(5)` system. This packaging method has been added in order for Message Queue to integrate with Sun GlassFish Enterprise Server 2.1.1.

Audit Logging Feature Reinstated Message Queue 3.7 provided an audit logging feature that was removed in Message Queue 4.0. This feature has been reinstated in Message Queue 4.4. For information about this feature, see "Audit Logging with the Solaris BSM Audit Log" in *Open Message Queue Administration Guide*.

New Features in Message Queue 4.3

Message Queue 4.3 was a minor release that included a number of feature enhancements and bug fixes. This section describes the new features included in this release:

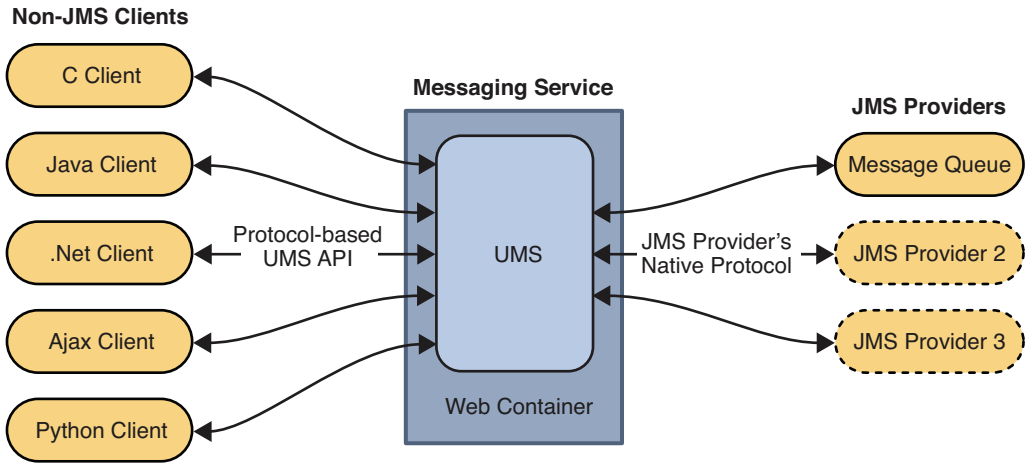
- Universal Message Service (UMS)
- AIX Platform Support
- New Zip-Based Installer
- Extended Platform Support
- Additional Enhancements

Universal Message Service (UMS)

Message Queue 4.3 introduces a new universal messaging service (UMS) and messaging API that provides access to Message Queue from any http-enabled device. As a result, almost any application can communicate with any other application and benefit from the reliability and guaranteed delivery of JMS messaging. In addition, the UMS provides enhanced scalability for JMS messaging, allowing the number of messaging clients to reach internet-scale proportions.

Architecture The basic UMS architecture is shown in the following figure:

Figure 1–1 UMS Architecture



The UMS, which runs in a web server, is language neutral and platform independent. The UMS serves as a gateway between any non-JMS client application and a JMS provider. It receives messages sent using the UMS API, transforms them into JMS messages, and produces them as persistent messages to destinations in the JMS provider by way of the provider's native protocol. Similarly, it retrieves messages from destinations in the JMS provider in a transacted session using AUTO_ACKNOWLEDGE mode, transforms them into text or SOAP messages, and sends the messages to non-JMS clients as requested by the clients through the UMS API.

The simple, language-independent, protocol-based UMS API supports both Web-based and non-Web-based applications, and can be used with both scripting and programming languages. The API is offered in two styles: a simple messaging API that uses a Representational State Transfer (REST)-style protocol, and an XML messaging API that embeds the protocol in a SOAP message header. In both cases, however, the API requires only a single http request to send or receive a message.

The simplicity and flexibility of the UMS API means that AJAX, .NET, Python, C, Java, and many other applications can send text message and/or SOAP (with attachment) messages to JMS destinations or receive messages from JMS destinations. For example, Python applications can communicate with .NET applications, iPhone can communicate with Java applications, and so forth.

For Message Queue 4.3, the UMS supports only Message Queue as a JMS provider.

Additional Features The UMS serves as more than the simple gateway described above. It supports stateful as well as stateless client sessions. If requested by the client, the UMS will maintain session state for the client application across multiple service requests. The UMS can use container-managed authentication, or be configured to authenticate clients with the Message Queue broker, or both. The UMS also supports transactions, enabling client applications to commit or roll back multiple service requests as a single atomic unit.

Because the UMS can support a large number of clients on a single connection to the Message Queue broker, it eases the load on the broker's connection services, allowing for maximum scalability. In addition, UMS capacity can be increased by horizontal scaling, allowing for internet-scale messaging loads.

On the client side, because of the simplicity of the protocol-based UMS API, no client libraries are required. As a result, the API can be extended in the future to implement additional JMS features without any need to upgrade client applications.

Using the UMS To use the UMS, you deploy the UMS into a web container that supports Servlet 2.4 or later specifications, start the Message Queue broker, create the appropriate destinations, and write a messaging application that uses the UMS API to send or receive messages.

The UMS `imqums.war` file, contained in the Message Queue 4.3 distribution, is installed in the following location, depending on platform:

You can rename the `.war` file as appropriate.

After you have deployed the `imqums.war` into a web container at `localhost:`*port*, you can find UMS at:

`http://localhost:`*port*`/imqums`

Otherwise you can find UMS as follows:

- For information on configuring the UMS, see `http://mq.java.net/4.4-content/imqums/config.html`.
- For of the UMS API, see `http://mq.java.net/4.4-content/imqums/protocol.html`.
- For programming examples in several languages, see `http://mq.java.net/4.4-content/imqums/examples/README.html`.

Supported Web Containers UMS is currently supported on the following web containers:

- Sun GlassFish Enterprise Server, Version 2.1 and Version 3 Prelude
- Tomcat, Versions 5.5 and 6.0

AIX Platform Support

Message Queue 4.3 provides AIX platform packages and an Installer for installing them).

The Message Queue AIX implementation supports the following software:

- AIX v 6.1 or higher (earlier versions of AIX are supported via the Unix/Java Only bundle)
- DB2 support
- IBM XL C/C++ Compiler V9.0
- JDK 1.5 or better

For installation instructions, see *AIX Installation* in *Sun Java System Message Queue 4.3 Installation Guide.*

On the AIX platform, Message Queue files are installed under a single Message Queue home directory, `IMQ_HOME`. `IMQ_HOME` denotes the directory *mqInstallHome*`/mq`, where *mqInstallHome* is the installation home directory you specify when installing the product (by default, *home-directory*`/MessageQueue`).

The resulting Message Queue directory structure is the same as that for the Windows platform (see the Windows section of "Distribution-Specific Locations of Message Queue Data" in *Open Message Queue Administration Guide*.)

Message Queue support for the AIX platform includes support for the Message Queue C-API. For instructions on building and compiling C applications on the AIX platform, see XREF.

New Zip-Based Installer

Message Queue 4.3 introduces a new installer for Zip-based distributions, as opposed to native package distributions. The installer is used to install the new Message Queue .zip distributions for the AIX platform.

The new installer extracts Message Queue `.zip` files to any directory for which you have write access (you do not need root privileges) and it also enables you to register your Message Queue installation with Sun Connection.

To minimize the size of download bundles, the Java Runtime is no longer be included in the zip-based distribution (most sites will already have it). As a result, the `installer` command requires that a JDK or JRE be specified, either by using the `JAVA_HOME` environment variable or by using the `-j` option on the command line, as follows:

`$ installer -j` *JDK/JRE-path*

where *JDK/JRE-path* is the path of the specified JDK or JRE.

Extended Platform Support

The following updated platform support will be certified for Message Queue 4.3:

- Oracle 11g
- Windows Server 2008

Additional Enhancements

The following additional enhancements are included in Message Queue 4.3:

- New Directory Structure on Windows Platform
- New Broker Properties
- JMX Administration API Enhancements
- Listing Durable Subscriptions for Wildcard Subscribers

New Directory Structure on Windows Platform The installed directory structure for Message Queue on the Windows platform has been modified from previous versions to match that of the AIX platform. This directory structure will be adopted as well by the Solaris and Linux platforms in the future, to facilitate multiple installations on single computer and automatic update of Message Queue through Sun Connection, a Sun-hosted service that helps you track, organize, and maintain Sun hardware and software (see Installer Support for Sun Connection Registration).

New Broker Properties The following new properties are available for configuring a broker:

Table 1–5 Broker Routing and Delivery Properties

Property	Type	Default Value	Description
`imq.transaction.producer.maxNumMsgs`	Integer	`1000`	The maximum number of messages that a producer can process in a single transaction. It is recommended that the value be less than 5000 to prevent the exhausting of resources.
`imq.transaction.consumer.maxNumMsgs`	Integer	`100`	The maximum number of messages that a consumer can process in a single transaction. It is recommended that the value be less than 1000 to prevent the exhausting of resources.
`imq.persist.jdbc.connection.limit`	Integer	`5`	The maximum number of connections that can be opened to the database.

JMX Administration API Enhancements A new attribute and composite data keys have been added to the JMX API as follows:

- A NextMessageID attribute has been added to the Destination Monitor MBean to provide the JMS message ID of the next message to be delivered to a consumer.
- A NextMessageID key for composite date has been added to the Consumer Manager Monitor MBean to provide the JMS message ID of the next message to be delivered to the consumer.
- A NumMsgsPending key for composite date has been added to the Consumer Manager Monitor MBean to provide the number of messages that have been dispatched to the consumer.

For more information see "Message Queue MBean Reference" in *Open Message Queue Developer's Guide for JMX Clients*.

Listing Durable Subscriptions for Wildcard Subscribers The command for listing durable subscriptions:

`list dur [-d` *topicName*`]`

has been enhanced to make specification of the topic name optional. If the topic is not specified, the command lists all durable subscriptions for all topics (including those with wildcard naming conventions)

New Features in Message Queue 4.2

Message Queue 4.2 was a minor release that included a number of new features, some feature enhancements, and bug fixes. This section describes the new features in the 4.2 release and provides further references for your use:

- Multiple Destinations for a Publisher or Subscriber

For information about features introduced in Message Queue 4.1 and 4.0, see New Features in Message Queue 4.1 and New Features in Message Queue 4.0, respectively.

Multiple Destinations for a Publisher or Subscriber

With Message Queue 4.2, a publisher can publish messages to multiple topic destinations and a subscriber can consume messages from multiple topic destinations. This capability is achieved by using a topic destination name that includes wildcard characters, representing multiple destinations. Using such symbolic names allows administrators to create additional topic destinations, as needed, consistent with the wildcard naming scheme. Publishers and subscribers automatically publish to and consume from the added destinations. (Wildcard topic subscribers are more common than publishers.)

Note: This feature does not apply to queue destinations.

The format of symbolic topic destination names and examples of their use is described in "Supported Topic Destination Names" in *Open Message Queue Administration Guide*.

Schema Validation of XML Payload Messages

This feature, introduced in Message Queue 4.2, enables validation of the content of a text (not object) XML message against an XML schema at the point the message is sent to the broker. The location of the XML schema (XSD) is specified as a property of a Message Queue destination. If no XSD location is specified, the DTD declaration within the XML document is used to perform DTD validation. (XSD validation, which includes data type and value range validation, is more rigorous than DTD validation.)

For information on the use of this feature, see "Schema Validation of XML Payload Messages" in *Open Message Queue Developer's Guide for Java Clients*.

C-API Support for Distributed Transactions

According to the X/Open distributed transaction model, support for distributed transactions relies upon a distributed transaction manager which tracks and manages operations performed by one or more resource managers. With Message Queue 4.2, the Message Queue C-API supports the XA interface (between a distributed transaction manager and Message Queue as a XA-compliant resource manager), allowing Message Queue C-API clients running in a distributed transaction processing environment (such as BEA Tuxedo) to participate in distributed transactions.

This distributed transaction support consists of the following new C-API functions (and new parameters and error codes) used to implement the XA interface specification:

```
MQGetXAConnection()
MQCreateXASession()
```

If a C-client application is to be used in the context of a distributed transaction, then it must obtain a connection by using MQGetXAConnection() and create a session for

producing and consuming messages by using MQCreateXASession(). The start, commit, and rollback, of any distributed transaction is managed through APIs provided by the distributed transaction manager.

For details of using the distributed transaction functions, see "Working With Distributed Transactions" in *Open Message Queue Developer's Guide for C Clients*.

Message Queue 4.2 provides programming examples based on the Tuxedo transaction manager. For information on the use of these sample programs, see "Distributed Transaction Sample Programs" in *Open Message Queue Developer's Guide for C Clients*.

> **Note:** The distributed transaction functionality is supported on Solaris, Linux, and Windows platforms, however, to date it has only been certified on the Solaris platform.

Installer Support for Sun Connection Registration

The Message Queue installer has been enhanced to allow for registration of Message Queue with Sun Connection, a Sun-hosted service that helps you track, organize, and maintain Sun hardware and software.

As part of Message Queue installation, you can choose to register Message Queue with Sun Connection. Information about the installed Message Queue, such as the release version, host name, operating system, installation date, and other such basic information is securely transmitted to the Sun Connection database. The Sun Connection inventory service can help you organize your Sun hardware and software, while the update service can inform you of the latest available security fixes, recommended updates, and feature enhancements.

For details of registering Message Queue with Sun Connection, see *Sun Java System Message Queue 4.3 Installation Guide*.

Support for MySQL Database

Message Queue 4.2 introduced support for MySQL database as a JDBC-based data store. MySQL Cluster Edition can be used as a JDBC database for a standalone broker, and MySQL Cluster Edition can be used as the highly-available shared data store needed for an enhanced broker cluster. For information on configuring Message Queue to use MySQL, see "Configuring a JDBC-Based Data Store" in *Open Message Queue Administration Guide* and also "Enhanced Broker Cluster Properties" in *Open Message Queue Administration Guide*.

Additional Enhancements

In addition to the features described above, Message Queue 4.2 included the following enhancements:

- **Remotely Produced Message Metrics**

  Message Queue 4.2 introduced new destination metrics that can be useful in monitoring destinations in a broker cluster. In a broker cluster, the messages stored in a given destination on a given broker in the cluster, consist of messages produced directly to the destination as well as messages sent to the destination from remote brokers in the cluster. In analyzing message routing and delivery in a broker cluster, it is sometimes helpful to know how many messages in a destination are local (locally produced) and how many are remote (remotely produced).

Two new physical destination metric quantities are included in Message Queue 4.2:

- `Num messages remote`, the current number of messages stored in memory and persistent store that were produced to a remote broker in a cluster, except for messages included in transactions.
- `Total Message bytes remote`, the current total size in bytes of messages stored in memory and persistent store that were produced to a remote broker in a cluster, except for messages included in transactions.

These new metric quantities are available through the `imqcmd list dst` and `imqcmd query dst` commands (see "Viewing Physical Destination Information" in *Open Message Queue Administration Guide*) and through new JMX attributes (see "Destination Monitor" in *Open Message Queue Developer's Guide for JMX Clients*).

- **Wildcard Producer and Wildcard Consumer Information**

  Information to support the use of wildcard characters in destination names (see Multiple Destinations for a Publisher or Subscriber) is provided through new monitoring data. For example, the number of wildcard producers or consumers associated with a destination are available through the `imqcmd query dst` command (see "Viewing Physical Destination Information" in *Open Message Queue Administration Guide*) and through new JMX attributes (see "Destination Monitor" in *Open Message Queue Developer's Guide for JMX Clients*). Also, wildcard information is available through the `ConsumerManager Monitor` and `ProducerManager Monitor` MBeans.

- **Support for DN Username Format for Client Authentication**

  Message Queue 4.2 introduced support for DN username format in client connection authentication against an LDAP user repository. The support involves the following new broker property (and value):

  ```
  imq.user_repository.ldap.usrformat=dn
  ```

  This property lets the broker authenticate a client user against an entry in an LDAP user repository by extracting from the DN username format the value of the attribute specified by the following property:

  ```
  imq.user_repository.ldap.uidattr
  ```

  The broker uses the value of the above attribute as the name of the user in access control operations.

  For example, if `imq.user_repository.ldap.uidattr=udi` and a client authentication username is in the format `udi=mquser,ou=People,dc=red,dc=sun,dc=com`, then "mquser" would be extracted for performing access control.

- **JAAS Authentication Enhancement**

  Message Queue 4.2 introduced JAAS authentication by IP address as well as by username.

New Features in Message Queue 4.1

Message Queue 4.1 was a minor release that included a number of new features, some feature enhancements, and bug fixes. This section describes the new features in the 4.1 release and provides further references for your use:

- High-Availability Broker Clusters

- JAAS Support
- Persistent Data Store Format Change
- Broker Environment Configuration
- Java ES Monitoring Framework Support
- Enhanced Transaction Management
- Fixed Ports for C Client Connections

For information about features introduced in Message Queue 4.0, see New Features in Message Queue 4.0.

High-Availability Broker Clusters

Message Queue 4.1 introduced a new, enhanced broker cluster. As compared to a conventional broker cluster, which provides only *messaging service* availability (if a broker fails, another broker is available to provide messaging service), the enhanced broker cluster also provides *data* availability (if a broker fails, its persistent messages and state data are available to another broker to use to take over message delivery).

The high-availability implementation introduced in Message Queue 4.1 uses a shared JDBC-based data store: instead of each broker in a broker cluster having its own persistent data store, all brokers in the cluster share the same JDBC-compliant database. If a particular broker fails, another broker within the cluster takes over message delivery for the failed broker. In doing so, the failover broker uses data and state information in the shared data store. Messaging clients of the failed broker reconnect to the failover broker, which provides uninterrupted messaging service.

The shared JDBC-based store used in the Message Queue 4.1 high-availability implementation must itself be highly available. If you do not have a highly available database or if uninterrupted message delivery is not important to you, you can continue to use conventional clusters, which provide service availability without data availability.

To configure a Message Queue 4.1 enhanced broker cluster, you specify the following broker properties for each broker in the cluster:

- *Cluster membership properties,* which specify that the broker is in an enhanced broker cluster, the ID of the cluster, and the ID of the broker within the cluster.
- *Highly available database properties*, which specify the persistent data model (JDBC), the name of the database vendor, and vendor-specific configuration properties.
- *Failure detection and failover properties*, which specify how broker failure is detected and handled using a failover broker.

To use the enhanced broker cluster implementation, you must do the following:

1. Install a highly available database.
2. Install the JDBC driver .jar file.
3. Create the database schema for the highly available persistent data store.
4. Set high-availability properties for each broker in the cluster.
5. Start each broker in the cluster.

For a conceptual discussion of enhanced broker clusters and how they compare to conventional clusters, see "Broker Clusters" in *Open Message Queue Technical Overview*. For procedural and reference information about enhanced broker clusters, see

"Configuring and Managing Broker Clusters" and "Cluster Configuration Properties" in *Open Message Queue Administration Guide*.

If you have been using a highly available database with Message Queue 4.0 and want to switch to an enhanced broker cluster, you can use the Database Manager utility (`imqdbmgr` to convert to a shared persistent data store. Also see Broker Clusters for more known issues and limitations.

JAAS Support

In addition to the file-based and LDAP-based built-in authentication mechanisms, Message Queue 4.1 introduced support for the Java Authentication and Authorization Service (JAAS), which allows you to plug an external authentication mechanism into the broker to authenticate Message Queue clients.

For a description of the information that a broker makes available to a JAAS-compliant authentication service and an explanation of how to configure the broker to use such a service, see "Using JAAS-Based Authentication" in *Open Message Queue Administration Guide*.

Persistent Data Store Format Change

Message Queue 4.1 changed the JDBC-based data store to support enhanced broker clusters. For this reason the format of the JDBC—based data store is increased to version 410. Format versions 350, 370, and 400 are automatically migrated to the 410 version.

Please note that the format of the file-based persistent data store remains at version 370 because no changes were made to it.

Broker Environment Configuration

The property `IMQ_DEFAULT_EXT_JARS` has been added to the Message Queue 4.1 environment configuration file, `imqenv.conf`. You can set this property to specify the path names of external .jar files to be included in `CLASSPATH` when the broker starts up. If you use this property to specify the location of external .jar files, you no longer need to copy these files to the `lib/ext` directory. External .jar files can refer to JDBC drivers or to JAAS login modules. The following sample poperty, specifies the location of JDBC drivers.

```
IMQ_DEFAULT_EXT_JARS=/opt/SUNWhadb4/lib/hadbjdbc4.jar:/opt/SUNWjavadb/derby.jar
```

Java ES Monitoring Framework Support

Message Queue 4.1 introduced support for the Sun Java Enterprise System (Java ES) Monitoring Framework, which allows Java ES components to be monitored using a common graphical interface. This interface is implemented by a web-based console called the Sun Java System Monitoring Console. Administrators can use the Console to view performance statistics, reate rules for automatic monitoring, and acknowledge alarms. If you are running Message Queue along with other Java ES components, you might find it more convenient to use a single interface to manage all of them.

For information on using the Java ES monitoring framework to monitor Message Queue, see XREF.

Enhanced Transaction Management

Previously, only transactions in a `PREPARED` state were allowed to be rolled back administratively. That is, if a session that was part of a distributed transaction did not

terminate gracefully, the transaction remained in a state that could not be cleaned up by an administrator. In Message Queue 4.1, you can now use the Command utility (`imqcmd`) to clean up (roll back) transactions that are in the following states: `STARTED`, `FAILED`, `INCOMPLETE`, `COMPLETE`, and `PREPARED`.

To help you determine whether a particular transaction can be rolled back (especially when it is not in a `PREPARED` state), the Command utility provides additional data as part of the`imqcmd query txn` output: it provides the connection id for the connection that started the transaction and specifies the time when the transaction was created. Using this information, an administrator can decide whether the transaction needs to be rolled back. In general, the administrator should avoid rolling back a transaction prematurely.

Fixed Ports for C Client Connections

In Message Queue 4.1, C clients, like Java clients, can now connect to a fixed broker port rather than to a port dynamically assigned by the broker's Port Mapper service. Fixed port connections are useful if you're trying to get through a firewall or if you need to bypass the Port Mapper service for some other reason.

To configure a fixed port connection you need to configure both the broker and the C client run time (both ends of the connection). For example, if you want to connect your client via `ssljms` to port 1756, you would do the following:

- On the client side, set the following properties:

  ```
  MQ_SERVICE_PORT_PROPERTY=1756

  MQ_CONNECTION_TYPE_PROPERTY=SSL
  ```

- On the broker side, set the `imq.`*serviceName.protocolType*`.port` property as follows:

  ```
  imq.ssljms.tls.port=1756
  ```

> **Note:** The `MQ_SERVICE_PORT_PROPERTY` connection property has been backported to Message Queue 3.7 Update 2.

New Features in Message Queue 4.0

Message Queue 4.0 was a minor release limited to supporting Application Server 9 PE. It included a few new features, some feature enhancements, and bug fixes. This section includes a description of new features in this release:

- Support for JMX Administration API
- Client Runtime Logging
- Connection Event Notification API
- Broker Administration Enhancements
- Displaying Information About a JDBC-Based Data Store
- JDBC Provider Support
- Persistent Data Store Format Changes
- Additional Message Properties
- SSL Support

Caution: One of the minor but potentially disruptive changes introduced with version 4.0 was the deprecation of the command-line option to specify a password. Henceforth, you must store all passwords in a file as described in Deprecated Password Option, or enter them when prompted.

Support for JMX Administration API

A new API was added in Message Queue 4.0 for configuring and monitoring Message Queue brokers in conformance with the Java Management Extensions (JMX) specification. Using this API, you can configure and monitor broker functions programmatically from within a Java application. In earlier versions of Message Queue, these functions were accessible only from the command line administration utilities or the Administration Console.

For more information see the *Open Message Queue Developer's Guide for JMX Clients*.

Client Runtime Logging

Message Queue 4.0 introduced support for client runtime logging of connection and session-related events.

Fore information regarding client runtime logging and how to configure it, see the Java Dev Guide pag 137.

Connection Event Notification API

Message Queue 4.0 introduced an event notification API that allows the client runtime to inform an application about changes in connection state. Connection event notifications allow a Message Queue client to listen for closure and re-connection events and to take appropriate action based on the notification type and the connection state. For example, when a failover occurs and the client is reconnected to another broker, an application might want to clean up its transaction state and proceed with a new transaction.

For information about connection events and how to create an event listener, see the Java Dev Guide, page 96.

Broker Administration Enhancements

In Message Queue 4.0, a new subcommand and several command options were added to the Command utility (`imqcmd`) to allow administrators to quiesce a broker, to shutdown a broker after a specified interval, to destroy a connection, or to set java system properties (for example, connection related properties).

- Quiescing a broker moves it into a quiet state, which allows messages to be drained before the broker is shut down or restarted. No new connections can be created to a broker that is being quiesced. To quiesce the broker, enter a command like the following.

  ```
  imqcmd quiesce bkr -b Wolfgang:1756
  ```

- To shut down the broker after a specified interval, enter a command like the following. (The time interval specifies the number of seconds to wait before the broker is shut down.)

  ```
  imqcmd shutdown bkr -b Hastings:1066 -time 90
  ```

  If you specify a time interval, the broker will log a message indicating when shutdown will occur. For example,

```
Shutting down the broker in 29 seconds (29996 milliseconds)
```

While the broker is waiting to shut down, its behavior is affected in the following ways.

- Administrative jms connections will continue to be accepted.
- No new jms connections will be accepted.
- Existing jms connections will continue to work.
- The broker will not be able to take over for any other broker in an enhanced broker cluster.
- The imqcmd utility will not block, it will send the request to shut down to the broker and return right away.

- To destroy a connection, enter a command like the following.

```
imqcmd destroy cxn -n 2691475382197166336
```

Use the command `imqcmd list cxn` or `imqcmd query cxn` to obtain the connection ID.

- To set a system property using `imqcmd`, use the new -D option. This is useful for setting or overriding JMS connection factory properties or connection-related java system properties. For example:

```
imqcmd list svc -secure -DimqSSLIsHostTrusted=true
imqcmd list svc -secure -Djavax.net.ssl.trustStore=/tmp/mytruststore
            -Djavax.net.ssl.trustStorePassword=mytrustword
```

For complete information about the syntax of the `imqcmd` command, see "Command Line Reference" in *Open Message Queue Administration Guide*.

Displaying Information About a JDBC-Based Data Store

In Message Queue 4.0 a new `query` subcommand was added to the Database Manager utility, `imqdbmgr`. This subcommand is used to display information about a JDBC-based data store, including the database version, the database user, and whether the database tables have been created.

The following is an example of the information displayed by the command.

```
imqdbmgr query

[04/Oct/2005:15:30:20 PDT] Using plugged-in persistent store:
        version=400
        brokerid=Mozart1756
        database connection url=jdbc:oracle:thin:@Xhome:1521:mqdb
        database user=scott
Running in standalone mode.
Database tables have already been created.
```

JDBC Provider Support

In Message Queue 4.0, Apache Derby Version 10.1.1 is now supported as a JDBC-based data store provider.

Persistent Data Store Format Changes

Message Queue 4.0 introduced changes to the JDBC-based data store for optimization and to support future enhancements. For this reason the format of the JDBC-based

data store was increased to version 400. Note that in Message Queue 4.0, the file-based data store version remains 370 because no changes were made to it.

Additional Message Properties

Message Queue 4.0 added two new properties which are set on all messages that are placed in the dead message queue.

- `JMS_SUN_DMQ_PRODUCING_BROKER` indicates the broker where the message was produced.
- `JMS_SUN_DMQ_DEAD_BROKER` indicates the broker who marked the message dead.

SSL Support

Starting with Message Queue 4.0, the default value for the client connection factory property `imqSSLIsHostTrusted` is `false`. If your application depends on the prior default value of `true`, you need to reconfigure and to set the property explicitly to `true`.

You might choose to trust the host when the broker is configured to use self-signed certificates. In this case, in addition to specifying that the connection should use an SSL-based connection service (using the imqConnectionType property), you should set the `imqSSLIsHostTrusted` property to true.

For example, to run client applications securely when the broker uses self-signed certificates, use a command like the following.

```
java -DimqConnectionType=TLS
      -DimqSSLIsHostTrusted=true ClientAppName
```

To use the Command utility (`imqcmd`) securely when the broker uses self-signed certificates, use a command like the following (for listing connector services).

```
imqcmd list svc -secure -DimqSSLIsHostTrusted=true
```