



# **Sun Java System Message Queue 4.0 Release Notes**



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

Part No: 819-5946

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

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

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

This distribution may include materials developed by third parties.

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

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

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

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

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

---

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

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

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

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

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

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

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

LA DOCUMENTATION EST FOURNIE "EN L'ÉTAT" ET TOUTES AUTRES CONDITIONS, DECLARATIONS ET GARANTIES EXPRESSES OU TACITES SONT FORMELLEMENT EXCLUES, DANS LA MESURE AUTORISÉE PAR LA LOI APPLICABLE, Y COMPRIS NOTAMMENT TOUTE GARANTIE IMPLICITE RELATIVE À LA QUALITÉ MARCHANDE, À L'APTITUDE À UNE UTILISATION PARTICULIÈRE OU À L'ABSENCE DE CONTREFAÇON.

# Contents

---

<b>1</b>	<b>Sun Java System Message Queue 4.0 Release Notes .....</b>	<b>5</b>
	Release Notes Revision History .....	6
	About Message Queue 4.0 .....	6
	What's New in This Release .....	6
	Hardware and Software Requirements .....	20
	Bugs Fixed in This Release .....	20
	Important Information .....	22
	Compatibility Issues .....	22
	Documentation Updates for Message Queue 4.0 .....	22
	Known Issues and Limitations .....	23
	Deprecated Password Option .....	23
	General Issues .....	24
	JMX Issues .....	25
	Administration/Configuration Issues .....	25
	Broker Issues .....	25
	Redistributable Files .....	26
	Accessibility Features for People With Disabilities .....	27
	How to Report Problems and Provide Feedback .....	27
	Sun Java System Software Forum .....	27
	Java Technology Forum .....	28
	Sun Welcomes Your Comments .....	28
	Additional Sun Resources .....	28





# Sun Java System Message Queue 4.0 Release Notes

---

Version 4.0

Part Number 819-5946

These release notes contain important information available at the time of release of Sun Java™ System Message Queue 4.0. New features and enhancements, known issues and limitations, and other information are addressed here. Read this document before you begin using Message Queue.

The most up-to-date version of these release notes can be found at the Sun Java System Message Queue documentation web site. 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 documentation.

These release notes contain the following sections:

- [“Release Notes Revision History” on page 6](#)
- [“About Message Queue 4.0” on page 6](#)
- [“Bugs Fixed in This Release” on page 20](#)
- [“Important Information” on page 22](#)
- [“Known Issues and Limitations” on page 23](#)
- [“Redistributable Files” on page 26](#)
- [“Accessibility Features for People With Disabilities” on page 27](#)
- [“How to Report Problems and Provide Feedback” on page 27](#)
- [“Sun Welcomes Your Comments” on page 28](#)
- [“Additional Sun Resources” on page 28](#)

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

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

# Release Notes Revision History

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

TABLE 1-1 Revision History

Date	Description of Changes
May 2006	Initial release of this Release Notes document.

## About Message Queue 4.0

Sun Java System Message Queue is a full-featured message service that provides reliable, asynchronous messaging conformant to the Java Messaging Specification (JMS) 1.1. In addition, Message Queue provides features that go beyond the JMS specification to meet the needs of large-scale enterprise deployments.

Message Queue 4.0 is a release limited to supporting Application Server 9 PE. It is a minor release that includes a few new features, minor enhancements, and bug fixes. This section includes the following information.

- [“What’s New in This Release” on page 6](#)
- [“Hardware and Software Requirements” on page 20](#)

## What’s New in This Release

Message Queue 4.0 includes the following new features:

- [“Interface Changes to the C-API and C Client Runtime” on page 7](#)
- [“Displaying Information About the Persistent Store” on page 8](#)
- [“Persistent Store Format Changes” on page 8](#)
- [“Broker Administration” on page 9](#)
- [“JDBC Persistence Support” on page 9](#)
- [“SSL Support” on page 9](#)
- [“JMX Support” on page 9](#)
- [“Client Runtime Logging” on page 14](#)
- [“Connection Event Notification” on page 18](#)

These are described in the following subsections.



**Caution** – One of the more minor but potentially disruptive changes introduced with version 4.0 is 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”](#) on page 23.

## Interface Changes to the C-API and C Client Runtime

Changes to the C-API were introduced with Release 3.7: they include a new function, a new message type, and a new connection property.

- New function: `MQGetDestinationName()`

```
MQGetDestinationName (const MQDestinationHandle destinationHandle,
                      MQString * destinationName);
```

Use this function to get the name of a destination. The returned `destinationName` is a copy that the caller is responsible for freeing by calling the `MQFreeString()` function.

### Parameters

*destinationHandle*      A handle to the destination whose name you want to know.

*destinationName*      The output parameter for the name.

This function is useful when using the reply-to pattern. You can use the `MQGetMessageReplyTo` function to obtain a handle to the destination where the message should be sent. You can then use the `MQGetDestinationName` to get the name of that destination. Having obtained the destination name, you can do message processing based on the name.

- New enumerated value: `MQ_MESSAGE`

The new `MQMessageType`, `MQ_MESSAGE`, allows C clients to exchange JMS messages of type Message with other Message Queue clients (both C and Java):

```
typedef enum _MQMessageType {MQ_TEXT_MESSAGE = 0,
                             MQ_BYTES_MESSAGE = 1,
                             MQ_MESSAGE = 3,
                             MQ_UNSUPPORTED_MESSAGE = 2} MQMessageType;
```

The `MQ_MESSAGE` type identifies messages that have a header and properties but no message body. You use the `MQCreateMessage()` function to create a message of this type.

- A new connection property, `MQ_UPDATE_RELEASE_PROPERTY`, that specifies the update release version for the installed version of Message Queue. Use the `MQGetMetaData()` function to obtain version information.

## Displaying Information About the Persistent Store

The query subcommand was added to the `imqdbmgr` command. Use this subcommand to display information about the persistent store, including the store version, the database user, and whether the database tables have been created.

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

```
dbmgr 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.
```

## Persistent Store Format Changes

Version 3.7 UR1 of Message Queue introduced two changes to the persistent store format to improve performance. One change is to the file store, the other is to the JDBC store.

- **Format of Transaction Data Persisted in File Store**

The format of transaction state information stored in the Message Queue file-based persistent store was changed to reduce disk I/O and to improve the performance of JMS transactions.

- **Oracle JDBC Store**

In previous versions of Message Queue, the store schema for Oracle used the `LONG RAW` data type to store message data. In Oracle 8, Oracle introduced the `BLOB` data types and deprecated the `LONG RAW` type. Message Queue 3.7 UR1 switched to the `BLOB` data type to improve performance and supportability.

Because these changes impact store compatibility, the store version for both the file store and the JDBC store was changed from 350 to 370 in version 3.7 UR1 of Message Queue.

Version 4.0 of Message Queue introduced changes to the JDBC store for optimization and to support future enhancements. For this reason the JDBC store version was increased to 400. Note that in Version 4.0, the file—based persistent store version remains 370 because no changes were made to it.

Message Queue 4.0 supports automatic conversion of the persistent store to the newest versions of the file-based and JDBC persistent stores. The first time `imqbrokerd` starts, if the utility detects an older store it will migrate the store to the new format, leaving the old store behind.

- File—based store versions 200 and 350 will be migrated to the 370 version format.
- JDBC store versions 350 and 370 will be migrated to the 400 version format. (If you need to upgrade a 200 store, you will need to step through an intermediate 3.5 or 3.6 release.)



If you should need to roll back this upgrade, you can uninstall Message Queue 4.0 and then reinstall the version you were previously running. Since the older copy of the store is left intact, the broker can run with the older copy of the store.

## Broker Administration

The Command utility (`imqcmd`) has added a subcommand and several options that allow administrators to quiesce the broker, to shutdown the broker after a specified interval, or to destroy a connection.

- Quiescing the 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
```

- 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.

For complete information about the syntax of the `imqcmd` command, see Chapter 14, “Command Line Reference,” in *Sun Java System Message Queue 3 2005Q4 Administration Guide*.

## JDBC Persistence Support

Apache Derby Version 10.1.1 is now supported as a JDBC-compliant persistent store provider.

## SSL Support

Starting with release 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`.

## JMX Support

A new API has been added 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 Message Queue client application. In earlier versions of Message Queue, these functions were accessible only from the command line or the Administration Console.

The API consists of a set of JMX *Managed Beans* (*MBeans*) for managing the following Message Queue–related resources:

- Message brokers
- Connection services
- Connections
- Destinations
- Message producers
- Message consumers
- Transactions
- Broker clusters
- Logging
- The Java Virtual Machine (JVM)

These MBeans provide *attributes* and *operations* for synchronously polling and manipulating the state of the underlying resources, as well as *notifications* that allow a client application to listen for and respond asynchronously to state changes as they occur. Using the JMX API, client applications can perform configuration and monitoring tasks like 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
- Get the current number of connections on a service
- Destroy a connection
- Create a destination
- Destroy a destination
- Enable or disable auto-creation of destinations
- Purge all messages from a destination
- Get the cumulative number of messages received by a destination since the broker was started
- Get the current state (running or paused) of a queue
- Get the current number of message producers for a topic
- Purge all messages from a durable subscriber
- Get the current JVM heap size

For an introduction to the JMX API and for complete reference information, see the *Sun Java System Message Queue 4.0 Developer's Guide for JMX Clients*.

## Broker Support: JMX-Related Properties

Several new broker properties have been added to support the JMX API (see [Table 1–2](#)). None of these properties can be set from the command line with the Message Queue Command utility (`imqcmd`). Instead, they can either be set with the `-D` option of the Broker utility (`imqbrokerd`) or edited by hand in the broker's instance configuration file (`config.properties`). In addition, some of

these properties (`imq.jmx.rmiregistry.start`, `imq.jmx.rmiregistry.use`, `imq.jmx.rmiregistry.port`) can be set with the new Broker utility options described in [Table 1–3](#). The table lists each option, specifies its type, and describes its use.

**TABLE 1–2** New Broker Properties for JMX Support

Property	Type	Description
<code>imq.jmx.rmiregistry.start</code>	Boolean	<p>Start RMI registry at broker startup?</p> <p>If <code>true</code>, the broker will start an RMI registry at the port specified by <code>imq.jmx.rmiregistry.port</code> and use it to store the RMI stub for JMX connectors. Note that the value of <code>imq.jmx.rmiregistry.use</code> is ignored in this case.</p> <p><b>Default value:</b> <code>false</code></p>
<code>imq.jmx.rmiregistry.use</code>	Boolean	<p>Use external RMI registry?</p> <p>Applies only if <code>imq.jmx.rmiregistry.start</code> is <code>false</code>.</p> <p>If <code>true</code>, the broker will use an external RMI registry at the port specified by <code>imq.jmx.rmiregistry.port</code> to store the RMI stub for JMX connectors. The external RMI registry must already be running at broker startup.</p> <p><b>Default value:</b> <code>false</code></p>
<code>imq.jmx.rmiregistry.port</code>	Integer	<p>Port number of RMI registry</p> <p>Applies only if <code>imq.jmx.rmiregistry.start</code> or <code>imq.jmx.rmiregistry.use</code> is <code>true</code>. JMX connectors can then be configured to use the RMI registry by including this port number in the URL path of their JMX service URLs.</p> <p><b>Default value:</b> <code>1099</code></p>
<code>imq.jmx.connector.list</code>	String	<p>Names of preconfigured JMX connectors, separated by commas</p> <p><b>Default value:</b> <code>jmxrmi,ssljmxrmi</code></p>
<code>imq.jmx.connector.activelist</code>	String	<p>Names of JMX connectors to be activated at broker startup, separated by commas</p> <p><b>Default value:</b> <code>jmxrmi</code></p>

TABLE 1–2 New Broker Properties for JMX Support (Continued)

Property	Type	Description
<code>imq.jmx.connector.connectorName.urlpath</code>	String	<p><i>urlPath</i> component of JMX service URL for connector <i>connectorName</i></p> <p>Useful in cases where the JMX service URL path must be set explicitly (such as when a shared external RMI registry is used).</p> <p><b>Default value:</b> If an RMI registry is used to store the RMI stub for JMX connectors (that is, if <code>imq.jmx.registry.start</code> or <code>imq.jmx.registry.use</code> is <code>true</code>):</p> <p style="text-align: center;"><i>/jndi/rmi://brokerHost:rmiPort /brokerHost/brokerPort/connectorName</i></p> <p>If an RMI registry is not used (the default case, <code>imq.jmx.registry.start</code> and <code>imq.jmx.registry.use</code> both <code>false</code>):</p> <p style="text-align: center;"><i>/stub/rmiStub</i></p> <p>where <i>rmiStub</i> is an encoded and serialized representation of the RMI stub itself</p>
<code>imq.jmx.connector.connectorName.useSSL</code>	Boolean	<p>Use Secure Socket Layer (SSL) for connector <i>connectorName</i>?</p> <p><b>Default value:</b> <code>false</code></p>
<code>imq.jmx.connector.connectorName.brokerHostTrusted</code>	Boolean	<p>Trust any certificate presented by broker for connector <i>connectorName</i>?</p> <p>Applies only when <code>imq.jmx.connector.connectorName.useSSL</code> is <code>true</code>.</p> <p>If <code>false</code>, the Message Queue client runtime will validate all certificates presented to it. Validation will fail if the signer of the certificate is not in the client's trust store.</p> <p>If <code>true</code>, validation of certificates is skipped. This can be useful, for instance, during software testing when a self-signed certificate is used.</p> <p><b>Default value:</b> <code>false</code></p>

The `imq.jmx.connector.list` property defines a set of named JMX connectors to be created at broker startup; `imq.jmx.connector.activelist` specifies which of these are to be activated. Each named connector then has its own set of properties:

```

imq.jmx.connector.connectorName.urlpath
imq.jmx.connector.connectorName.useSSL
imq.jmx.connector.connectorName.brokerHostTrusted

```

By default, two JMX connectors are created, named `jmxrmi` and `ssljmxrmi`; the first is configured not to use SSL encryption (`imq.jmx.connector.jmxrmi.useSSL = false`, the second to use it (`imq.jmx.connector.ssljmxrmi.useSSL = true`). By default, only the `jmxrmi` connector is activated at broker startup; see “[SSL Support for JMX Clients](#)” on page 13 for information on how to activate the `ssljmxrmi` connector for secure communications.

For convenience, new options ([Table 1–3](#)) are also added to the command-line Broker utility (`imqbrokerd`) to control the usage, startup, and port for the RMI registry. The use and effects of these options are the same as those of the equivalent broker properties, as described in [Table 1–2](#). The table lists each option, specifies its equivalent broker property, and describes its use.

**TABLE 1–3** New Broker Utility Options for JMX Support

Option	Equivalent Broker Property	Description
<code>-startRmiRegistry</code>	<code>imq.jmx.rmiregistry.start</code>	Start RMI registry at broker startup?
<code>-useRmiRegistry</code>	<code>imq.jmx.rmiregistry.use</code>	Use external RMI registry?
<code>-rmiRegistryPort</code>	<code>imq.jmx.rmiregistry.port</code>	Port number of RMI registry

A new subcommand ([Table 1–4](#)) is added to the command-line Command utility (`imqcmd`) for listing the JMX service URLs of JMX connectors created and started at broker startup. This information is needed by JMX clients that do not use the Message Queue convenience class `AdminConnectionFactory` to obtain their JMX connectors, and can also be used for managing or monitoring Message Queue via a generic JMX browser such as the Java Monitoring and Management Console (`jconsole`).

**TABLE 1–4** New Command Utility Subcommand

Subcommand	Description
<code>list jmx</code>	List JMX service URLs of JMX connectors

## SSL Support for JMX Clients

As mentioned above, a Message Queue message broker is configured by default for insecure communication using the preconfigured JMX connector `jmxrmi`. Applications wishing to use the Secure Socket Layer (SSL) for secure communication must activate the alternate, secure JMX connector, `ssljmxrmi`. This requires the following steps:

1. Obtain and install a signed certificate in the same way as for the `ssljms`, `ssladmin`, or `cluster` connection service, as described in the *Message Queue Administration Guide*.
2. Install the root certification authority certificate in the trust store if necessary.
3. Add the `ssljmxrmi` connector to the list of JMX connectors to be activated at broker startup:

```
imq.jmx.connector.activelist=jmxrmi,ssljmxrmi
```

4. Start the broker with the Message Queue Broker utility (`imqbrokerd`), either passing it the key-store password in a password file or typing it from the command line when prompted.
5. By default, the `ssljmxrmi` connector (or any other SSL-based connector) is configured to validate all broker SSL certificates presented to it. To avoid this validation (for instance, when using self-signed certificates during software testing), set the broker property `imq.jmx.connector.ssljmxrmi.brokerHostTrusted` to `true`.

On the client side, the administrator connection factory (`AdminConnectionFactory`) must be configured with a URL specifying `ssljmxrmi` as the preferred connector:

```
AdminConnectionFactory acf = new AdminConnectionFactory();  
acf.setProperty(AdminConnectionFactory.imqAddress, "mq://myhost:7676/ssljmxrmi");
```

If needed, use the system properties `javax.net.ssl.trustStore` and `javax.net.ssl.trustStorePassword` to point the JMX client to the trust store.

## Client Runtime Logging

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

JDK 1.4 (and above) includes the `java.util.logging` library. This library implements a standard logger interface that can be used for application-specific logging.

The Message Queue client runtime uses the Java Logging API to implement its logging functions. You can use all the J2SE 1.4 logging facilities to configure logging activities. For example, an application can use the following Java logging facilities to configure how the Message Queue client runtime outputs its logging information:

- Logging Handlers
- Logging Filters
- Logging Formatters
- Logging Level

For more information about the Java Logging API, please see the Java Logging Overview at <http://java.sun.com/j2se/1.4.2/docs/guide/util/logging/overview.html>

## Logging Name Spaces, Levels, and Activities

The Message Queue provider defines a set of logging name spaces associated with logging levels and logging activities that allow Message Queue clients to log connection and session events when a logging configuration is appropriately set.

The root logging name space for the Message Queue client runtime is defined as `javax.jms`. All loggers in the Message Queue client runtime use this name as the parent name space.

The logging levels used for the Message Queue client runtime are the same as those defined in the `java.util.logging.Level` class. This class defines seven standard log levels and two additional settings that you can use to turn logging on and off.

OFF	Turns off logging.
SEVERE	Highest priority, highest value. Application-defined.
WARNING	Application-defined.
INFO	Application-defined.
CONFIG	Application-defined
FINE	Application-defined.
FINER	Application-defined.
FINEST	Lowest priority, lowest value. Application-defined.
ALL	Enables logging of all messages.

In general, exceptions and errors that occur in the Message Queue client runtime are logged by the logger with the `javax.jms` name space.

- Exceptions thrown from the JVM and caught by the client runtime, such as `IOException`, are logged by the logger with the logging name space `javax.jms` at level `WARNING`.
- JMS exceptions thrown from the client runtime, such as `IllegalStateException`, are logged by the logger with the logging name space `javax.jms` at level `FINER`.
- Errors thrown from the JVM and caught by the client runtime, such as `OutOfMemoryError`, are logged by the logger with the logging name space `javax.jms` at level `SEVERE`.

The following tables list the events that can be logged and the log level that must be set to log events for JMS connections and for sessions.

The following table describes log levels and events for connections.

**TABLE 1-5** Log Levels and Events for `javax.jms.connection` Name Space

Log Level	Events
FINE	Connection created
FINE	Connection started
FINE	Connection closed
FINE	Connection broken
FINE	Connection reconnected
FINER	Miscellaneous connection activities such as <code>setClientID</code>

**TABLE 1-5** Log Levels and Events for `javax.jms.connection` Name Space (Continued)

Log Level	Events
FINEST	Messages, acknowledgments, Message Queue action and control messages (like committing a transaction)

For sessions, the following information is recorded in the log record.

- Each log record for a message delivered to a consumer includes `ConnectionID`, `SessionID`, and `ConsumerID`.
- Each log record for a message sent by a producer includes `ConnectionID`, `SessionID`, `ProducerID`, and destination name.

The table below describes log levels and events for sessions.

**TABLE 1-6** Log Levels and Events for `javax.jms.session` Name Space

Log Level	Event
FINE	Session created
FINE	Session closed
FINE	Producer created
FINE	Consumer created
FINE	Destination created
FINER	Miscellaneous session activities such as committing a session.
FINEST	Messages produced and consumed. (Message properties and bodies are not logged in the log records.)

By default, the output log level is inherited from the JRE in which the application is running. Check the `JRE_DIRECTORY/lib/logging.properties` file to determine what that level is.

You can configure logging programmatically or by using configuration files, and you can control the scope within which logging takes place. The following sections describe these possibilities.

## Using the JRE Logging Configuration File

The following example shows how you set logging name spaces and levels in the `JRE_DIRECTORY/lib/logging.properties` file, which is used to set the log level for the Java runtime environment. All applications using this JRE will have the same logging configuration. The sample configuration below sets the logging level to `INFO` for the `javax.jms.connection` name space and specifies that output be written to `java.util.logging.ConsoleHandler`.



```

#logging.properties file.
# "handlers" specifies a comma separated list of log Handler
# classes. These handlers will be installed during VM startup.
# Note that these classes must be on the system classpath.
# By default we only configure a ConsoleHandler, which will only
# show messages at the INFO and above levels.

handlers= java.util.logging.ConsoleHandler

# Default global logging level.
# This specifies which kinds of events are logged across
# all loggers. For any given facility this global level
# can be overridden by a facility-specific level.
# Note that the ConsoleHandler also has a separate level
# setting to limit messages printed to the console.

.level= INFO

# Limit the messages that are printed on the console to INFO and above.

java.util.logging.ConsoleHandler.level = INFO
java.util.logging.ConsoleHandler.formatter =
    java.util.logging.SimpleFormatter

# The logger with javax.jms.connection name space will write
# Level.INFO messages to its output handler(s). In this configuration
# the output handler is set to java.util.logging.ConsoleHandler.

javax.jms.connection.level = INFO

```

## Using a Logging Configuration File for a Specific Application

You can also define a logging configuration file from the java command line that you use to run an application. The application will use the configuration defined in the specified logging file. In the following example, `configFile` uses the same format as defined in the `JRE_DIRECTORY/lib/logging.properties` file.

```
java -Djava.util.logging.config.file=configFile MQApplication
```

## Setting the Logging Configuration Programmatically

The following code uses the `java.util.logging` API to log connection events by changing the `javax.jms.connection` name space log level to `FINE`. You can include such code in your application to set logging configuration programmatically.

```

import java.util.logging.*;
//construct a file handler and output to the mq.log file
//in the system's temp directory.

```

```
Handler fh = new FileHandler("%t/mq.log");
fh.setLevel (Level.FINE);

//Get Logger for "javax.jms.connection" domain.

Logger logger = Logger.getLogger("javax.jms.connection");
logger.addHandler (fh);

//javax.jms.connection logger would log activities
//with level FINE and above.

logger.setLevel (Level.FINE);
```

## Connection Event Notification

Connection event notifications allow a Message Queue client to listen for closure and reconnection 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.

If the Message Queue provider detects a serious problem with a connection, it calls the connection object's registered exception listener. It does this by calling the listener's `onException` method, and passing it a `JMSEException` argument describing the problem. In the same way, the Message Queue provider offers an event notification API that allows the client runtime to inform the application about connection state changes. The notification API is defined by the following elements:

- The `com.sun.messaging.jms.notification` package, which defines the event listener and the notification event objects.
- The `com.sun.messaging.jms.Connection` interface, which defines extensions to the `javax.jms.Connection` interface.

The following sections describe the events that can trigger notification and explain how you can create an event listener.

## Connection Events

The following table lists and describes the events that can be returned by the event listener.

Note that the JMS exception listener is not called when a connection event occurs. The exception listener is only called if the client runtime has exhausted its reconnection attempts. The client runtime always calls the event listener before the exception listener.

TABLE 1–7 Notification Events

Event Type	Meaning
<code>ConnectionClosingEvent</code>	The Message Queue client runtime generates this event when it receives a notification from the broker that a connection is about to be closed due to a shutdown requested by the administrator.
<code>ConnectionClosedEvent</code>	<p>The Message Queue client runtime generates this event when a connection is closed due to a broker error or when it is closed due to a shutdown or restart requested by the administrator.</p> <p>When an event listener receives a <code>ConnectionClosedEvent</code>, the application can use the <code>getEventCode()</code> method of the received event to get an event code that specifies the cause for closure.</p>
<code>ConnectionReconnectedEvent</code>	<p>The Message Queue client runtime has reconnected to a broker. This could be the same broker to which the client was previously connected or a different broker.</p> <p>An application can use the <code>getBrokerAddress</code> method of the received event to get the address of the broker to which it has been reconnected.</p>
<code>ConnectionReconnectFailedEvent</code>	<p>The Message Queue client runtime has failed to reconnect to a broker. Each time a reconnect attempt fails, the runtime generates a new event and delivers it to the event listener.</p> <p>The JMS exception listener is not called when a connection event occurs. It is only called if the client runtime has exhausted its reconnection attempts. The client runtime always calls the event listener before the exception listener.</p>

## Creating an Event Listener

The following code example illustrates how you set a connection event listener. Whenever a connection event occurs, the event listener's `onEvent` method will be invoked by the client runtime

```
//create an MQ connection factory.

com.sun.messaging.ConnectionFactory factory =
    new com.sun.messaging.ConnectionFactory();

//create an MQ connection.

com.sun.messaging.jms.Connection connection =
```

```
(com.sun.messaging.jms.Connection )factory.createConnection();

//construct an MQ event listener. The listener implements
//com.sun.messaging.jms.notification.EventListener interface.

com.sun.messaging.jms.notification.EventListener eListener =
    new ApplicationEventListener();

//set event listener to the MQ connection.

connection.setEventListener ( eListener );
```

**Event Listener Examples**

In this example, an application chooses to have its event listener log the connection event to the application’s logging system:

```
public class ApplicationEventListener implements
    com.sun.messaging.jms.notification.EventListener {

public void onEvent ( com.sun.messaging.jms.notification.Event connEvent ) {
    log (connEvent);
}
private void log ( com.sun.messaging.jms.notification.Event connEvent ) {
    String eventCode = connEvent.getEventCode();
    String eventMessage = connEvent.getEventMessage();
    //write event information to the output stream.
}
}
```

**Hardware and Software Requirements**

Hardware and software requirements are specified in “Hardware and Software Requirements” in *Sun Java System Application Server Platform Edition 9 Release Notes*.

**Bugs Fixed in This Release**

The following tables describes the bugs fixed in Message Queue 4.0.

TABLE 1–8 Fixed Bugs in Message Queue 4.0

Bug Number	Description
4986481	In Message Queue 3.5, callingSession.recover could hang in auto-reconnect mode.

TABLE 1–8 Fixed Bugs in Message Queue 4.0 (Continued)

Bug Number	Description
4987325	Redelivered flag was set to false for redelivered messages after calling <code>Session.recover</code> .
6157073	Change new connection message to include the number of connections on the service in addition to the total number of connections.
6193884	Message Queue outputs garbage message to syslog in non-C locale.
6196233	Message selection using <code>JMSMessageID</code> doesn't work.
6251450	<code>ConcurrentModificationException</code> on <code>connectList</code> during cluster shutdown.
6252763	<code>java.nio.BufferOverflowException</code> in <code>java.nio.HeapByteBuffer.putLong/Int</code> .
6260076	First message published after startup is slow with Oracle storage.
6264003	The queue browser shows uncommitted messages.
6260814	Selector processing on <code>JMSXUserID</code> always evaluates to false.
6264003	The queue browser shows uncommitted messages.
6271876	Connection Flow Control does not work properly when closing a consumer with unconsumed messages.
6279833	Message Queue should not allow two brokers to use the same jdbc tables.
6293053	Master broker does not start up correctly if the system's IP address is changed, unless the store is cleared (using <code>-reset store</code> .)
6294767	Message Queue broker needs to set <code>SO_REUSEADDR</code> on the network sockets it opens.
6304949	Unable to set <code>ClientID</code> property for <code>TopicConnectionFactory</code> .
6307056	The txn log is a performance bottleneck.
6320138	Message Queue C API lacks ability to determine the name of a queue from a reply-to header.
6320325	The broker sometimes picks up JDK 1.4 before JDK 1.5 on Solaris even if both versions are installed.
6321117	Multibroker cluster initialization throws <code>java.lang.NullPointerException</code> .
6330053	The jms client throws <code>java.lang.NoClassDefFoundError</code> when committing a transaction from the subscriber.
6340250	Support MESSAGE type in C-API.
6351293	Add Support for Apache Derby database.

## Important Information

This section contains the latest information that is not contained in the core product documentation. This section covers the following topics:

- “Compatibility Issues” on page 22
- “Documentation Updates for Message Queue 4.0” on page 22

## Compatibility Issues

This section covers compatibility issues in Message Queue 4.0.

### Interface Stability

Sun Java System Message Queue uses many interfaces that may change over time. Chapter 21 in *Sun Java System Message Queue 3 2005Q4 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.

### Issues Related to the Next Major Release of Message Queue

The next major release of Message Queue may introduce changes that make your clients incompatible with that release. This information is provided now to allow you to prepare for these changes.

- The locations of individual files installed as part of Sun Java System Message Queue might change. This could break existing applications that depend on the current location of certain Message Queue files.
- 3.5 and earlier brokers may no longer be able to operate in a cluster with newer brokers.
- In future releases, Message Queue clients may not be able to use JDK versions that are earlier than 1.3.

## Documentation Updates for Message Queue 4.0

Other than this *Release Notes* document, Message Queue 4.0 includes only one new document: *Sun Java™ System Message Queue 4.0 Developer's Guide for JMX Clients*.

The Message Queue documentation that was published for Message Queue 3.6 SP3, 2005Q4, is up to date with respect to the needs of Application Server 9 PE clients. This documentation set is available at the following location.

<http://docs.sun.com/app/docs/coll/1307.1>

# Known Issues and Limitations

This section contains a list of the known issues with Message Queue 4.0. The following product areas are covered:

- “Deprecated Password Option” on page 23
- “General Issues” on page 24
- “JMX Issues” on page 25
- “Administration/Configuration Issues” on page 25
- “Broker Issues” on page 25

For a list of current bugs, their status, and workarounds, Java Developer Connection™ members should see the Bug Parade page on the Java Developer Connection web site. 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.

<http://bugs.sun.com/bugdatabase/index.jsp>

---

**Note** – Java Developer Connection membership is free but requires registration. Details on how to become a Java Developer Connection member are provided on Sun’s “For Developers” web page.

---

To report a new bug or submit a feature request, send mail to [imq-feedback@sun.com](mailto:imq-feedback@sun.com).

## 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. You must furnish passwords as follows.

1. Set the appropriate property to the desired password value in a file used to store only passwords. Use the following syntax to specify passwords in the password file.

*PasswordPropertyName=MyPassword*

2. Pass the name of the password file using the `-passfile` option.

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.password` property to specify this password.

- 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 to `abracadabra`.

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

You can configure the broker to use the password file you create in one of the following ways.

- Set the following properties in the broker's `config.properties` file.

```
imq.passfile.enabled=true
imq.passfile.dirpath=MyFileDirectory
imq.passfile.name=MyPassfileName
```

- Use the `-passfile` option of the `imqbrokerd` command.

```
imqbrokerd -passfile MyPassfileName
```

## General Issues

This section covers general issues in Message Queue 4.0. Some of these were introduced with previous Message Queue versions.

The following issues affect both editions of the Message Queue 4.0 product.

- SOAP clients. Previously the SAAJ 1.2 implementation jar used to refer to `mail.jar` and `mail.jar` did not need to be in `CLASSPATH`. In SAAJ 1.3 this reference was removed; thus, Message Queue clients must put `mail.jar` explicitly in `CLASSPATH`.
- In Message Queue 4.0, the example broker configuration for using an LDAP server as a user repository is provided in the comment area in the `config.properties` file, and the LDAP user repository example in the `default.properties` file has been commented out.

If you previously relied on any property value in the example LDAP user repository properties specified in the `default.properties` file, your JMS application client will receive a security exception when attempting to create a JMS connection. This will happen after you upgrade to Message Queue 4.0.

When your JMS client tries to make a connection to the Message Queue 4.0 broker, you will get a error in the broker log and your JMS client will receive the following exception:

```
SecurityException.
20/Aug/2004:11:16:41 PDT] ERROR [B4064]: Ldap repository ldap property
.uidattr not defined for authentication type
basic:com.sun.messaging.jmq.auth.LoginException:
[B4064]: Ldap repository ldap property .uidattr not defined
for authentication type basic
```

*Workaround* Set the broker property `imq.user_repository.ldap.uidattr` following the instructions in Chapter 8 in *Sun Java System Message Queue 3 2005Q4 Administration Guide*.



## JMX Issues

The following issues pertain to the use of the JMX API.

- On the Windows platform, the `getTransactionInfo` method of the Transaction Manager Monitor MBean returns transaction information that has incorrect transaction creation time (*Bug ID 6393359*).  
*Workaround* Use the `getTransactionInfoByID` method of the Transaction Manager Monitor MBean instead.
- The `getActiveConsumerIDs` method of the Destination Monitor MBean throws an exception when used with topic destinations (*Bug ID 6397184*).  
*Workaround* Use the `getConsumerIDs` method of the Destination Monitor MBean instead.

## Administration/Configuration Issues

The following issues pertain to administration and configuration of Message Queue

- The `imqadmin` and `imqobjmgr` utilities throw an error when the `CLASSPATH` contains double quotes on Windows machines (*Bug ID 5060769*).  
*Workaround* You can ignore this error message; the broker correctly handles notifying consumers of any error. This error does not affect the reliability of the system.
- The `-javahome` option in all Solaris and Windows scripts does not work if the value provided contains a space (*Bug ID 4683029*).  
The `javahome` option is used by Message Queue commands and utilities to specify an alternate Java 2 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 1.4`  
Solaris: `/work/java 1.4`  
*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 destination. Note that the client application will always get all the messages on the queue. Thus, the `imqQueueBrowserMaxMessagesPerRetrieve` attribute affects how the queued messages are chunked, to be delivered to the client runtime (fewer large chunks, or more smaller chunks), but it does not affect the total messages browsed. Changing the value of this attribute might impact performance, but it will not result in the client application getting more or less data (*Bug ID 6387631*).

## Broker Issues

The following issues affect the Message Queue broker.

- When using Ctrl-C to shut down broker, transactions may be cleaned up after store is closed (*Bug ID 4934446*).

The broker may show errors with the following reason “Store method accessed after the store is closed.” if the broker is shut down while messages or transactions are processed.

*Workaround* You can ignore this error message; the broker correctly handles notifying consumers of any error. This error does not affect the reliability of the system.

- Broker becomes inaccessible when persistent store opens too many destinations. (*Bug ID 4953354*).

*Workaround* This condition is caused by the broker reaching the system open-file descriptor limit. On Solaris and Linux use the `ulimit` command to increase the file descriptor limit.

- Consumers are orphaned when a destination is destroyed (*Bug ID 5060787*).

Active consumers are orphaned when a destination is destroyed. Once the consumers have been orphaned, they will no longer receive messages (even if the destination is recreated).

*Workaround* There is no workaround for this problem.

- Message selection using `JMSMessageID` doesn’t work (*Bug ID 6196233*).

*Workaround* Change the selector from the following expression:

```
JMSMessageID = "ID:message-id-string"
```

to the following expression:

```
JMSMessageID IN ( 'ID:message-id-string', 'message-id-string' )
```

- Message Queue A queue browser shows uncommitted messages (*Bug ID 6264003*).

When browsing the contents of a queue, messages that were produced in a transaction but are not yet committed may appear in the queue browser enumeration.

*Workaround* There is no workaround for this problem.

- Connection is dropped for a broker in a cluster (*Bug ID 6377527*).

One reason this can happen is that the broker address (whose connection is dropped) is resolved to be the loopback IP address (127.0.0.1).

*Workaround* Make sure that the broker address does not resolve to the loopback IP address.

## Redistributable Files

Sun Java System Message Queue 4.0 contains the following set of files which you may use and freely distribute in binary form:

<code>fscontext.jar</code>	<code>jms.jar</code>
<code>imq.jar</code>	<code>jndi.jar</code>
<code>imqxm.jar</code>	<code>jnet.jar</code>
<code>jaas.jar</code>	<code>jsse.jar</code>
<code>jcrt.jar</code>	<code>ldap.jar</code>

ldapbpjar  
 libmqcrt.so (HPUX)  
 libmqcrt.so (UNIX)

mqcrt1.dll (Windows)  
 providerutil.jar

In addition, you can also redistribute the LICENSE and COPYRIGHT files.

## Accessibility Features for People With Disabilities

To obtain accessibility features that have been released since the publishing of this media, consult Section 508 product assessments (available from Sun upon request) to determine which versions are best suited for deploying accessible solutions. Updated versions of applications can be found at <http://sun.com/software/javaenterprisesystem/get.html>.

For information on Sun's commitment to accessibility, visit <http://sun.com/access>.

## How to Report Problems and Provide Feedback

If you have problems with Sun Java System Message Queue, contact Sun customer support using one of the following mechanisms:

- Sun Software Support services online at <http://www.sun.com/service/sunone/software>.  
 This site has links to the Knowledge Base, Online Support Center, and ProductTracker, as well as to maintenance programs and support contact numbers.
- The telephone dispatch number associated with your maintenance contract.

So that we can best assist you in resolving problems, please have the following information available when you contact support:

- Description of the problem, including the situation where the problem occurs and its impact on your operation.
- Machine type, operating system version, and product version, including any patches and other software that might be affecting the problem.
- Detailed steps on the methods you have used to reproduce the problem.
- Any error logs or core dumps.

## Sun Java System Software Forum

There is a Sun Java System Message Queue forum available at the following location:

<http://swforum.sun.com/jive/forum.jspa?forumID=24>

We welcome your participation.

## Java Technology Forum

There is a JMS forum in the Java Technology Forums that might be of interest.

<http://forum.java.sun.com>

## Sun Welcomes Your Comments

Sun is interested in improving its documentation and welcomes your comments and suggestions.

To share your comments, go to <http://docs.sun.com> and click Send Comments. In the online form, provide the document title and part number. The part number is a seven-digit or nine-digit number that can be found on the title page of the book or at the top of the document. For example, the title of this book is Sun Java System Message Queue 4.0 Release Notes, and the part number is 819-5946.

## Additional Sun Resources

Useful Sun Java System information can be found at the following Internet locations:

- Documentation  
<http://docs.sun.com/prod/java.sys>
- Professional Services  
<http://www.sun.com/service/sunps/sunone>
- Software Products and Service  
<http://www.sun.com/software>
- Software Support Services  
<http://www.sun.com/service/sunone/software>
- Support and Knowledge Base  
<http://www.sun.com/service/support/software>
- Sun Support and Training Services  
<http://training.sun.com>
- Consulting and Professional Services  
<http://www.sun.com/service/sunps/sunone>
- Developer Information  
<http://developers.sun.com>
- Sun Developer Support Services  
<http://www.sun.com/developers/support>

- Software Training  
<http://www.sun.com/software/training>

