Release Notes for Sun™ ONE Message Queue

Version 3.0

Updated June, 2002

These release notes contain important information available at the time of release of Version 3.0 of Sun ONE Message Queue (MQ). New features and enhancements, known limitations and problems, technical notes, and other information are addressed here. Read this document before you begin using MQ 3.0.

The most up-to-date version of these release notes can be found at the Sun ONE documentation web site: `http://docs.iplanet.com/docs/manuals/`. Check the web site after installing your software, and then periodically thereafter, to view the most up-to-date version of these notes.

These release notes contain the following sections:

Java Message Service (JMS) Compliance

MQ 3.0 is designed to be compliant with the Java Message Service (JMS) 1.1 specification. Because an official Compatibility Test Suite (CTS) for JMS has not yet been released, we are unable to provide verification of JMS 1.1 compliance.

Known bugs related to JMS compliance, and their workarounds, are listed in the "Known Bugs" section of this document.

MQ Documentation Updates

The following MQ 3.0 documents have been updated from Version 2.0 of the product—formerly called iPlanet Message Queue (iMQ). These updated documents can be found at the Sun ONE documentation web site: `http://docs.iplanet.com/docs/manuals/`.

Installation Guide

The MQ 3.0 product includes an updated MQ *Installation Guide*.

Administrator's Guide

The MQ *Administrator's Guide* has been updated to include new MQ 3.0 features (see "What's New in MQ 3.0" on page 3).

Developer's Guide

The MQ *Developer's Guide* has been updated to include new MQ 3.0 features (see "What's New in MQ 3.0" on page 3).

What's New in MQ 3.0

The MQ 3.0 product includes a number of changes to Version 2.0 of the product—iMQ 2.0 (and iMQ 2.0, Service Pack 1).

Notable among these changes is that the product is now available in two editions: a Platform Edition and an Enterprise Edition.

Platform Edition. Provides basic JMS support, and is best suited to small-scare deployments and development environments

Enterprise Edition. Provides HTTP/HTTPS support, enhanced scalability, and security features, and is best suited to large-scale deployments.

(See the introduction to the MQ *Administrator's Guide* or the MQ *Developer's Guide* for more information on these editions.)

The descriptions, below, of changes in the MQ 3.0 product are grouped according to whether they apply to both editions or to the Enterprise Edition only.

Both Enterprise and Platform Editions

- Support for distributed transactions

  MQ now supports the JTA XA Resource API, meaning that production and consumption of messages can be part of a larger distributed transaction involving other resource managers, such as database managers (see Chapter 1 of the MQ *Developer's Guide*). This feature is also supported with administrative tools for managing transactions (see Table 6-12 of the MQ *Administrator's Guide*). Programming information and examples are not yet available in the MQ 3.0 release product.

- Support for JMS 1.1

  MQ now supports the added features of the JMS 1.1 specification, which provides a simplified approach to JMS client programming as compared to JMS 1.0.2. In particular, a JMS client can perform both point-to-point and publish/subscribe messaging over the same connection and within the same session, and can include both queues and topics in the same transaction.

In short, a JMS client developer need not make a choice between the separate point-to-point and publish/subscribe programming domains of JMS 1.0.2, opting instead for the simpler, unified domain approach of JMS 1.1. This is the preferred approach, however the JMS 1.1 specification continues to support the separate JMS 1.0.2 programming domains. (In fact, the example applications included with the MQ product as well as the code examples provided in the MQ *Developer's Guide* all use the separate JMS 1.0.2 programming domains.)

NOTE Developers of applications that run in the Sun ONE Application Server environment are limited to using the JMS 1.0.2 API. This is because the Sun ONE Application Server complies with the J2EE 1.3 specification, which supports only JMS 1.0.2. This means that any JMS messaging performed in servlets and EJBs (including message-driven beans) must be based on the domain-specific JMS APIs.

- Support for SOAP messaging using JAXM

  Supports creation and delivery of messages that conform to the Simple Object Access Protocol (SOAP) specification, using JAXM—the Java API for XML Messaging. SOAP allows for the exchange of structured XML data between peers in a distributed environment. MQ also supports the delivery of SOAP messages *via* JMS messaging. See the MQ *Developer's Guide* for more information.

- Improvements in persistent store

  MQ now provides more flexibility in balancing disk space and performance when using the built-in persistent store (see Table 2-5 of the MQ *Administrator's Guide*). Also, administrators now have the option of removing only messages or durable subscriptions from the persistent store when restarting a broker (see `reset` option in Table 5-2 of the MQ *Administrator's Guide*).

- Overriding JMS message header fields

  MQ now allows an administrator to better control message server resources by overriding the JMS message header fields that specify message persistence, priority, and expiration (see Table 4-4 of the MQ *Developer's Guide*).

- Improved management of durable subscriptions

  MQ now supports the purging of all messages for a specified durable subscription (see Table 6-11 of the MQ *Administrator's Guide*).

- New hostname configuration property

  MQ now supports more than one network interface card on a computer by letting administrators choose which hostname will be used by MQ connection services (see Table 2-3 of the MQ *Administrator's Guide*).

- Updating the default queue delivery policy

  MQ now allows an administrator to update the default delivery policy set for a queue destination (see Table 6-10 of the MQ *Administrator's Guide*).

- Support for Java 2 Platform, Standard Edition (J2SE) 1.4

  The broker and MQ administration tools are now supported on the Java Runtime Environment (JRE) 1.4, and JMS clients are now supported on the Java Software Development Kit (JDK) 1.4.

- New file system layout on Solaris

  The installed directory structure of MQ 3.0 on Solaris has been changed to conform to general file system standards for the platform. MQ 3.0 files are no longer installed under a single root installation directory, but are dispersed to standard locations in the Solaris file system.

Enterprise Edition Only

- Support for secure HTTP (HTTPS)

  MQ now supports secure messaging over HTTP (see Appendix B of the MQ *Administrator's Guide*). This new connection service provides for encryption of messages from message producer through to message consumer (that is, from JMS client, through HTTPS tunnel servlet, to broker, and *visa versa*).

- Increased client connection capacity

  MQ now provides a threadpool sharing option that can increase the number of client connections that can be made to an MQ broker (see Chapter 2 of the MQ *Administrator's Guide*).

Compatibility Issues

Due to changes made to improve features, MQ 3.0 is generally not compatible with iMQ 2.0. In particular, there are a number of issues that you might need to address when upgrading from iMQ 2.0 to MQ 3.0:

- Broker Compatibility
- Administered Object Compatibility
- Administration Tool Compatibility
- Client Compatibility

Broker Compatibility

An MQ 3.0 broker will not inter-operate with an iMQ 2.0 broker due to changes in broker properties and in the persistent store schema. However, some iMQ 2.0 data is compatible with MQ 3.0, as shown in Table 1, and can be preserved when upgrading to MQ 3.0. When upgrading from iMQ 2.0 to MQ 3.0, you should consider the following:

- You can copy iMQ 2.0 `config.properties` files to another location and, in most cases, consult the property settings they contain when you configure MQ 3.0 brokers.
- Any persistent iMQ 2.0 data—messages, destinations, durable subscriptions—cannot be re-used. In particular, you will need to re-create iMQ 2.0 destinations in your MQ 3.0 brokers.
- You can continue to use iMQ 2.0 user repository and access control properties files after installing MQ 3.0. The MQ 3.0 installer does not overwrite these files. However, you will have to move them to the appropriate MQ 3.0 location (see Appendix D of MQ *Administrator's Guide*).

Table 1 Compatibility of MQ 3.0 with iMQ 2.0 Data

iMQ 2.0 Data Category	Location of iMQ 2.0 Data	Compatibility with MQ 3.0
Broker properties	`IMQ_VARHOME/stores/`*brokerName*`/props/config.properties`	Incompatible; do not use.
Persistent store (messages, destinations, durable subscriptions, transactions)	`IMQ_VARHOME/stores/`*brokerName*`/filestore/` or JDBC-accessible data store	Incompatible; do not use.
Administered objects	local directory or LDAP server	Compatible; can use and/or convert to 3.0.

Table 1 Compatibility of MQ 3.0 with iMQ 2.0 Data *(Continued)*

iMQ 2.0 Data Category	Location of iMQ 2.0 Data	Compatibility with MQ 3.0
Security: user repositories	`IMQ_VARHOME/security/passwd` or LDAP server	Compatible. Move to following location: `IMQ_HOME/etc/passwd` (`/etc/imq/passwd` on Solaris)
Security: access control file	`IMQ_VARHOME/security/accesscontrol.properties`	Compatible. Move to following location: `IMQ_HOME/etc/...` (`/etc/imq/...` on Solaris)

Administered Object Compatibility

MQ 3.0 administered objects have been enhanced with new attributes and iMQ 2.0 attributes have been renamed. Therefore, when upgrading from iMQ 2.0 to MQ 3.0, you should consider the following:

- You can use the same object store and administered objects that you created in iMQ 2.0; however, it is best to upgrade your administered objects after installing MQ 3.0. The Administration Console (`imqadmin`) and the ObjectManager command line utility (`imqobjmgr`), when performing an update operation, will convert iMQ 2.0 administered objects into MQ 3.0 administered objects.
- The MQ 3.0 client runtime will look up and instantiate iMQ 2.0 administered objects by converting them into local MQ 3.0 administered objects, but this will *not* convert iMQ 2.0 administered objects in the object store into MQ 3.0 administered objects.
- JMS clients (applications and/or components) that directly instantiate administered objects—that is, that are JMS provider-dependent—need to be rewritten to accommodate new administered object attribute names (see Chapter 4 and Appendix A of the MQ *Developer's Guide* for information on administered object attributes).
- Scripts that start JMS clients and which set administered object attribute values using command line options need to be rewritten to accommodate the new administered object attribute names (see Chapter 4 and Appendix A of the MQ *Developer's Guide* for information on administered object attributes).

Administration Tool Compatibility

Because of the renaming of many files and directories (specifically to replace the string "jmq" with "imq"), all MQ 3.0 command line utilities, broker properties, administered object attributes, and internal file names have changed. Therefore, when upgrading from iMQ 2.0 to MQ 3.0, you should consider the following:

- Any scripts that use command line utilities (`imqbrokerd`, `imqcmd`, `imqobjmgr`, and so forth) need to be edited to replace the old commands with the newly-named commands. Note, especially, that the `jmqbroker` command is now `imqbrokerd`.
- The Administration Console (`imqadmin`) allows you to manage several brokers and/or object stores concurrently, and saves the list of managed entities that are displayed in the navigational pane on the left side of the screen. Thus each time you launch the Console, the list of managed entities is redisplayed. The name of the directory in which user settings for the iMQ 2.0 Administration Console were stored has changed for MQ 3.0. If you wish to preserve the old Console settings when upgrading from iMQ 2.0 to MQ 3.0, you need to change the name of the directory where the `brokerlist.properties` and `objstorelist.properties` files are stored from *`$HOME/`*`.jmq/admin` to *`$HOME/`*`.imq/admin`, where *`$HOME`* is the Console user's home directory.

Client Compatibility

When upgrading from iMQ 2.0 to MQ 3.0, you should consider the following:

- An MQ 3.0 broker will support the iMQ 2.0 client runtime (but without additional MQ 3.0 capabilities), but an iMQ 2.0 broker will *not* support the MQ 3.0 client runtime.
- JMS clients built on JDK 1.2, 1.3, or 1.4 can inter-operate with a broker running JRE 1.4. However, clients that use a secure (SSL-based) connection to a broker will require additional JSSE and JNDI libraries if they are not built on JDK 1.4 (which includes these libraries).
- The JMS 1.1 API (supported by MQ 3.0) clarifies the behavior of the `Message.acknowledge()` method, used to acknowledge message consumption in `CLIENT_ACKNOWLEDGE` sessions. This might require you to modify existing JMS clients.

  This `Message.acknowledge()` method now acknowledges *all* messages consumed in the session at the time the method is called. This change in behavior from the 1.0.2 API (supported by iMQ 2.0) is illustrated in the following example: suppose a client consumes four messages from a queue in the same session, say A, B, C, and D in that order, and all were consumed before the client calls the acknowledge method on message C.

  - In 1.0.2, only messages A, B, and C, would get acknowledged since D was consumed after message C.
  - In 1.1, all the messages (including D) are acknowledged since they were all consumed.

The acknowledgement is independent of the order in which messages are consumed, so long as they are consumed in the same session; or stated another way, the message on which the `acknowledge()` method is called no longer determines which messages get acknowledged.

- The JMS 1.1 API (supported by MQ 3.0) clarifies the use of the client identification used to keep track of durable subscriptions. This might require you to modify existing JMS clients.

  In iMQ 2.0, the behavior was to automatically set the ClientID to the local IP address of the client if a durable subscription was created without explicitly setting a ClientID value. In MQ 3.0, the behavior is to throw an exception if a durable subscription is created without explicitly setting a ClientID value. In other words a ClientID value must always be set—either in client code or using an attribute of the connection factory object—when durable subscriptions and durable connection consumers are used.

- When using message selectors in iMQ 2.0, a workaround was necessary to accommodate a bug that has been fixed in MQ 3.0. This might require you to modify existing JMS clients.

  In iMQ 2.0, if a string literal contained multi-byte characters, you had to use a double escape on Unicode characters (for example, `selector = “property = ‘\\u033e\\u033f’”`). In MQ 3.0, the normal representation for Unicode characters can be used (for example, `selector = “property = ‘\u033e\u033f’”`).

Known Limitations

Limitations shown in this section are grouped according to whether they apply to both Enterprise and Platform Editions of MQ 3.0 or to the Enterprise Edition only.

Both Enterprise and Platform Editions

- Windows platforms set limits to the number of simultaneous connections to a broker, in accordance with the maximum value of the backlog size. Backlog is the buffer for connections in the TCP stack—the number of simultaneous connections cannot exceed the backlog size. For example, Windows 2000 Professional limits the backlog to 5, and Windows 2000 Server limits the backlog to 200.
- Automatic reconnection of JMS clients to brokers is limited to connections where the client-side state can be fully restored on the broker upon reconnection. In all other cases, the connection Exception handler will get called and the client has to manually restore state.

  Clients using any of the following objects will have to explicitly restore state: temporary destinations, transacted sessions, `CLIENT_ACKNOWLEDGE` sessions, or `ConnectionConsumer` objects.
- Messages for different message consumers on a single connection are sent to the consuming client without regard to the precedence of the message consumers and their sessions. This means that a message consumer with a large number of pending messages in one session can adversely affect the performance of other sessions on the same connection. Message consumers that are expected to have very different message throughput levels should use different connections.
- You cannot edit a broker’s instance configuration file without having started the broker instance at least once. This is because the `config.properties` file does not exist until the broker instance is first started. To configure a broker to use pluggable persistence or to set other configuration properties, run the broker once (with the instance name that should be used to create the broker) to create the `config.properties` file:

  `IMQ_VARHOME/instances/`*brokerName*`/props/config.properties`
  (`/var/imq/instances/`*brokerName*`/props/config.properties` on Solaris)

  Once the `config.properties` file has been created, edit the file to add any configuration property values and then restart the broker.

- Due to an error, the `.class` files were omitted from the example application directories in the MQ product:

  `IMQ_HOME/demo/` (`/usr/demo/imq/` on Solaris)

  Should you need to compile these example applications, please follow the instructions in Chapter 2 of the MQ *Developer's Guide*.

Enterprise Edition Only

- The broker's shared thread pool model does not work on Windows platforms (due to a bug in JRE 1.4).
- Only fully-connected broker clusters are supported in this release. This means that every broker in a cluster must communicate directly with every other broker in the cluster. If you are connecting brokers using the `imqbrokerd -cluster` command line argument, be careful to ensure that all brokers in the cluster are included.
- If a Master Broker is not used in a broker cluster, persistent information stored by a broker being added to the cluster is not propagated to other brokers in the cluster.
- A connection service using SSL is currently limited to supporting only self-signed server certificates, that is, host-trusted mode. The connection configuration property `imqSSLIsHostTrusted` is set to true by default.
- MQ 3.0 is supported on Linux 7.1, but Sun ONE Web Server 6.0 SP2—a servlet provider of the kind required for the HTTP tunneling feature—is not supported on Linux 7.1.

  If you encounter problems running Web Server 6.0 SP2, on Linux 7.1, you can run the Web Server on an officially supported platform; that is, on a host different from that running the broker.
- When a JMS client using the HTTP transport terminates abruptly (for example, using `Ctrl-C`) the broker takes approximately one minute before the client connection and all the associated resources are released.

  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 “Resource in conflict” exception. This is not a real problem; it's 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.

Known Bugs

This section contains a listing of the more important bugs known at the time of the MQ 3.0 release.

For a list of current bugs, their status, and workarounds, Java Developer Connection (TM) 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 not all MQ bugs are listed here, it is a good starting place if you want to know whether a problem has been reported.

The relevant page is:

```
http://developer.java.sun.com/developer/index.html
```

NOTE Java Developer Connection membership is free but does require registration for access. 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`.

Table 2 Bug Descriptions

Bug Number	Details
4428745	A client connecting to a broker over HTTP may not receive notification that the broker has been shut down if that shutdown happens while the web server is down. The HTTP servlet maintains connections to the broker across webserver restarts. This prevents a client from realizing that the broker has shutdown until after both the webserver and the broker have been restarted. **Workaround**: You should make sure the broker is restarted promptly if there are HTTP clients. You can also set the client's `ConnectionFactory` property `imqAckTimeout` to a non-zero value to limit the timeout when using HTTP tunneling; this will help in situations where the HTTP client waits for replies from the broker.

Table 2 Bug Descriptions *(Continued)*

Bug Number	Details
4430941	`imqadmin/imqobjmgr`: Will not list objects of unknown type An object store is used to store MQ administered objects. It is accessed using the Java Naming and Directory Interface (JNDI). This API or interface can be used to store any kind of objects (not just MQ administered objects). Additionally, LDAP Directory Servers generally allow you to store various types of objects. It is then possible for an object store to contain a variety of objects—some of which are not relevant to MQ. The MQ administration tools (`imqobjmgr` and `imqadmin`) currently only list or display MQ administered objects. This may prevent administrators from realizing that there are other objects located in the object store. This issue may be important when the administrator attempts to add a new object to the object store. This is because it is possible that the lookup name specified is already in use by an object (that is not listed or displayed by the administration tools), causing an error. The administration tools provided with the LDAP server should provide you with the means to view all the objects in the object store. This will be addressed in a future release. **Workaround**: None.
4431924	`imqadmin`: modal dialogs can get into deadlock situation The Administration Console uses dialogs that are application modal. Most of these dialogs are brought up explicitly by interacting with the graphical user interface, for example, by selecting the Add Brokers menu item. On the other hand, some of these dialogs appear as a result of a lost broker connection. It is possible for multiple modal dialogs to be visible at the same time. When this happens, the Administration Console is locked. You will not be able to dismiss either modal dialog using the Close button. **Workaround**: Dismiss the top most dialog using the window manager controls i.e. - the 'X' button at the upper right corner on Windows - the 'Close' window manager menu item on Solaris
4449354	In extremely rare cases, calling the methods `Connection.stop`, `Connection.start`, and `Connection.close` at the same time as calling the methods `Session.recover` and `Session.rollback` (in separate threads) may result in an unexpected message redelivery order. **Workaround**: make sure your calls to the `Connection`… and `Session`… methods specified above are serialized either by using the same thread or by using synchronization.

Table 2 Bug Descriptions *(Continued)*

Bug Number	Details
4487650	The wrong port number for a service may be seen after the port is updated on Linux. The port number for a service can not be dynamically updated on Linux (because of a JDK bug). When you attempt to update the service using `imqcmd`, the command will fail with the following error: [B3109]: Cannot update service port number dynamically. The change will take effect after a broker restart. However, any future queries of the service will show the future port number (which will take affect after restart), not the actual port number of the service. **Workaround**: Restart the broker after changing the port number.
4487661	On Linux, changing the portmapper port will not immediately take affect A JDK bug on Linux prevents the broker from receiving immediate notification that the old socket used by the portmapper has been closed. Once a connection is attempted to the original port, the broker will received notification that the old port is closed and correctly configure the system. Until this notification is received, the new port will not respond to requests. **Workaround**: After changing the portmapper port, query the broker on the old port number, for example: `imqcmd query bkr -b host:`*old broker port* After this query, the broker will respond on the new port.
4635816	The Print Dialog from the Help Viewer in the Administration Console does not work when the Help Viewer is launched from a modal dialog on Solaris. **Workaround**: Dismiss the modal dialog from which the Help Viewer was launched. This will enable the Print Dialog and make it usable.
4679837	Client sometimes throws `JMSException` on connection.`close()` when TLS is used as transport. This problem is related to an SSL problem in JDK1.4 and bug 4688051 in the client. **Workaround**: Either ignore the Exception encountered when closing the connection or use JRE 1.2 or 1.3 on the client if you encounter this problem.
4683326	Locking protocol times out under heavy load. Heavy network traffic with large messages can occasionally clog the cluster connections. If you are running queue receiver or durable subscription clients with broker clusters, the increased latency can sometimes cause locking protocol timeout errors. As a result the client may get a "Resource in conflict" exception while trying to create consumers. Normally these problems can be avoided by using a higher speed connection. However if that's not possible you can also set the following broker configuration property to increase the locking protocol timeout - `imq.cluster.locktimeout` (value in seconds)

Table 2 Bug Descriptions *(Continued)*

Bug Number	Details
4685101	Broker shuts down itself under heavy load on Linux... When the broker gets close to exhausting the JVM heap space used by Java objects it uses various techniques such as flow control and message swapping to free the memory. Under extreme circumstances it even closes client connections in order to free the memory and reduce the message inflow. However if the maximum Java heap space is configured incorrectly the broker can continue to grow the Java heap space externally until the entire system runs out of memory. This can result in unpredictable broker crashes/shutdown. It can also affect the behavior of other applications and services running on the system. This problem can be avoided by configuring a sensible Java heap size limit using the `-Xmx` Java command line argument. In general it is a good idea to evaluate the normal and peak system memory footprints and configure applications accordingly.
4685329	Two-broker durable test failures. Running a broker cluster without a master broker (`imq.cluster.masterbroker`) can lead to unpredictable failures especially with durable subscription clients. The master broker is necessary to ensure the consistency of the persistent state information maintained by the clustered brokers. Please see the MQ *Administrator's Guide* for more details.
4687290	Broker shuts down itself in Windows If the broker does not have sufficient memory to handle the incoming message flow, it can get out of memory exceptions. Normally broker tries to recover from such errors. However if the attempt to free memory fails, in some cases the broker has no choice but to bail out. This situation can be avoided by allocating sufficiently large JVM heap using the `-Xmx` java command line argument.
4688051	Client can get an `EOFException` when rapidly creating and closing connections to the broker. **Workaround**: None. The exception is printed to the console and can be ignored. No message loss occurs.
4689962	The output of various admin utilities is neatly aligned in columns and in some cases, bordered with dashes ("-"). However, in the Japanese locale, the alignment is sometimes off and the borders are too short. **Workaround**: None
4694340	On Redhat Linux 7.1, the Java VM can sometimes crash with a SIGSEGV. This is JDK bug Id 4629175. This may be more likely to happen when you terminate the broker or client using a Ctrl-C. **Workaround**: Broker: Use the next version of the JRE which fixes this problem when it becomes available. Client: Use either the 1.2 or 1.3 JRE on Linux

Table 2 Bug Descriptions *(Continued)*

Bug Number	Details
4694971	Standalone JTS/XA transaction completion may fail ocassionally, even after a normal, error-free XAConnection.close(). **Workaround**: In most cases, completing the transaction, by calling the `commit()` or `rollback()` method on the TransactionManager before closing the connection using `XAConnection.close()` avoids the problem.
4696361	When the print button in the Help Viewer of the Administration Console is selected, the print dialog does not come up. **Workaround**: None
4700851	Client does not clean up local transaction after ending an XA transaction When an XA transaction is ended, a local transaction is started with the broker that isn't cleaned up until the connection is closed. This can cause broker memory to grow on long-lived connections that use distributed transaction connections. This will be fixed in a future release. **Workaround**: None
4701982	The Administration Console cannot be launched on Solaris or Linux if the CLASSPATH environment variable is set A `java.lang.NoClassDefFoundError` will be seen when the `imqadmin` script is invoked. This is due to a typo in the `imqadmin` script. **Workaround**: Unset the CLASSPATH variable.
4702152	For messages acknowledged as part of a consumer-side transaction, broker holds on to some unnecessary state information for each acknowledged message until the consumer is closed. This may cause the broker to grow in size if a long running transacted consumer receives a large number of messages. **Workaround**: Use `CLIENT_ACKNOWELDGE` instead of transactions, or periodically close the long-running consumer.
4704186	Corrections in the example applications `README` file (`demo/jms/README`) 1. On line 502: "AsyncTopicExample" should be "AsynchTopicExample" 2. On line 724: Should mention that the <systemid_url> command line option refers to the URL that contains the DTD for the <xml_filename> command line option. **Workaround**: As stated above.

Bugs Fixed in 3.0

Below is a short description of the most important bugs fixed in MQ 3.0. For more details about any of these bugs you can view the complete report at the Java Developer Connection site:

```
http://developer.java.sun.com/developer/bugParade
```

Table 3 Bugs Fixed in MQ 3.0

Bug Number	Description
4407510	The `imqcmd` and `imqadmin` utilities do not currently display temporary destinations in a broker.
4407510	`imqcmd`: should display temporary destinations
4407729	Broker should be able to bind to particular address on multi-home hosts
4431383	Message selection does not work properly if the selector string contains string literals that contain Japanese or other multi-byte characters.
4475983	Storing new destination with > 10,000k destinations is extremely slow
4477085	Service state not kept after restart
4478682	`imqcmd`, `imqadmin`: current # mesgs / bytes should be available when querying broker
4482742	`imqcmd`, `imqadmin`: should provide a way to set default flavor on autocreated queues
4487218	Broker needs a way to reset only messages and durable subscriptions, leaving destinations
4488580	`imqcmd/imqobjmgr: -v` does not show patch information on Windows
4495379	`imqcmd`: When you create a queue and topic with the same name, only one is listed
4513764	TCP services do not allow specific IP addresses to be specified
4523001	Destroying a temporary destination results in a null pointer exception
4554766	Temporary destination consumer creation fails in certain cases
4614357	Attempt to change ClientID doesn't throw `IllegalStateException`
4620654	Two connections creating same destination at same time may fail w/ server error
4624078	`Message.acknowledge()` doesn't acknowledge all consumed messages in session
4625363	`MessageConsumer.close()` discards acks in transacted sessions
4627901	NullPtrException seen when execute `jmqcmd list dur`
4630229	A client ID must not be set automatically, if it has not been set in the ConnectionFactory
4635583	`imqcmd`: need to add support for SSL as transport to broker

Functionality Marked as Optional in JMS

The JMS specification indicates certain items that are optional-- each JMS provider (vendor) chooses whether or not to implement them. The MQ product handling of each of these optional items is indicated below:

Table 4 Optional JMS Functionality

Section in JMS Specification	Description and MQ Handling
3.4.3 JMSMessageID	“Since message ID’s take some effort to create and increase a message’s size, some JMS providers may be able to optimize message overhead if they are given a hint that message ID is not used by an application. JMS Message Producer provides a hint to disable message ID.” **MQ implementation**: Product does not disable Message ID generation (any setDisableMessageID() call in MessageProducer is ignored). All messages will contain a valid MessageID value.
3.4.12 Overriding Message Header Fields	“JMS does not define specifically how an administrator overrides these header field values. A JMS provider is not required to support this administrative option.” **MQ implementation**: The MQ product supports administrative override of the values in message header fields through configuration of connection factory administered objects.
3.5.9 JMS Defined Properties	“JMS Reserves the ’JMSX’ Property name prefix for JMS defined properties.” “Unless noted otherwise, support for these properties is optional.” **MQ implementation**: The JMSX properties defined by the JMS 1.0.2 specification are supported in the MQ product.
3.5.10 Provider-specific Properties	“JMS reserves the ’`JMS_<vendor_name>`’ property name prefix for provider-specific properties.” **MQ implementation**: The purpose of the provider-specific properties is to provide special features needed to support JMS use with provider-native clients. They should not be used for JMS to JMS messaging. MQ 3.0 does not use provider-specific properties.
4.4.8 Distributed Transactions	“JMS does not require that a provider support distributed transactions.” **MQ implementation**: Distributed transactions are supported in this release of the MQ product.

Table 4 Optional JMS Functionality *(Continued)*

Section in JMS Specification	Description and MQ Handling
4.4.9 Multiple Sessions	"For PTP <point-to-point distribution model>, JMS does not specify the semantics of concurrent QueueReceivers for the same queue; however, JMS does not prohibit a provider from supporting this." See section 5.8 of the JMS specification for more information. **MQ implementation**: The MQ implementation supports three queue delivery policies: Failover, Round Robin, and Single (default). For more information, refer to the MQ *Administrator's Guide*.

Technical Notes

This section contains short write-ups on the following topics:

System Clock Settings

When using an MQ system, you should be careful to synchronize system clocks and avoid setting them backward.

Synchronization Recommended

It is recommended that you synchronize the clocks on all hosts interacting with the MQ system. This is particularly important if you are using message expiration (TimeToLive). Failure to synchronize the hosts' clocks may result in TimeToLive not working as expected (messages may not be delivered). You should synchronize clocks before starting any brokers.

Solaris. You can issue the `rdate` command on a local host to synchronize with remote host. (You must be superuser--that is, root--to run this command.) For example, the following command synchronizes the local host (call it Host 2) with remote host Host1:

```
# rdate Host1
```

Linux. The command is similar to Solaris, but you must provide the -s option:

```
# rdate -s Host1
```

Windows. you can issue the net command with the time subcommand to synchronize your local host with a remote host. For example, the following command synchronizes the local host (call it Host 2) with remote host Host1:

```
net time \\Host1 /set
```

Avoid Setting System Clocks Backwards

You should avoid setting the system clock backwards on systems running an MQ broker. MQ uses timestamps to help identify internal objects such as transactions and durable subscriptions. If the system clock is set backwards it is theoretically possible that a duplicate internal identifier can be generated. The broker attempts to compensate for this by introducing some randomness to identifiers and by detecting clock shift when running, but if the system clock is shifted backwards by a significant amount when a broker is not running, then there is a slight risk of identifier duplication.

If you need to set the system clock backwards on a system running a broker by more than a few seconds, it is recommended that you either do it when there are no transactions or durable subscriptions, or do it when the broker is not running, then wait the amount of time you have shifted the clock before bringing the broker back up.

But the ideal approach is to synchronize clocks before starting up any brokers, and then use an appropriate techinque to ensure that clocks don't drift significantly after deployment.

OS-Defined Connection Limitations on Clients and Brokers

On the Solaris and Linux platforms, the shell in which the client or broker is running places a soft limit on the number of file descriptors that a client can use. In the MQ system, each connection a client makes, or each connection a broker accepts, uses one of these file descriptors. As a result, you cannot have a broker or client running with more than 256 connections on Solaris or 1024 on Linux without changing this limit. (The number is actually slightly lower than that due to file descriptors that are used for other purposes, such as for file-based persistence.)

To change this limit, see the `ulimit` man page or the instructions under "Increasing File Descriptors to Improve File-based Persistence Performance," below. The limit needs to be changed in each shell in which a client or broker will be executing.

Increasing File Descriptors to Improve File-based Persistence Performance

On the Solaris and Linux platforms, the speed of storing messages in the default file-based persistence is affected by the number of file descriptors available for use by the file store. (Windows does not have a file descriptor limit.) A large number of descriptors will allow the system to process large numbers of persistent messages faster.

To improve performance for performance testing or deployment, administrators should increase the maximum number of file descriptors available to an application (in this case, the broker process) and then increase the size of the shared file descriptor pool used by the broker by updating the value of the property:

```
imq.persist.file.message.fdpool.limit
```

The value of this property must be less than the maximum number of file descriptors available on your system.

On Solaris, for example, you can increase the file descriptor limits using the `ulimit` command. Processes inherit system limits from their parent (login) shell. On Solaris, there is a "hard" limit and a "soft" limit. For a non-root user, the number of file descriptors for an application cannot exceed the soft limit, which, in turn, cannot exceed the hard limit.

To check the current file descriptor limits:

Hard limit: `$ ulimit -Hn`

Soft limit: `$ ulimit -n`

To change the file descriptor limits for "root" user:

```
# ulimit -Hn unlimited
# ulimit -n unlimited
```

After this, any process created from this shell will be able to open unlimited file descriptors. So it is safe to run the `imqbroker` command at this point.

To change the file descriptor limit for non-root user:

$ ulimit -Hn *number1*

$ ulimit -n *number2*

where *number1* is less than 1024, and *number2* is less than *number1*.

If 1024 is not enough, you have the following options:

- Run the broker as root.
- Write some "setuid" program to increase the ulimit value before running the broker. (Note - such programs pose tremendous security risk. Highly discouraged.)
- Tune the rlim_fd_max parameter in the `/etc/`system file and reboot the system.

Securing Persistent Data

The broker uses a persistent store that can contain, among other information, message files that are being temporarily stored. Since these messages might contain proprietary information, it is recommended that the data store be secured against unauthorized access.

A broker can use either the built-in or plugged-in persistence.

Built-in Persistent Store

A broker using built-in persistence writes persistent data to a flat file data store located at:

`IMQ_VARHOME/instances/`*brokerName*`/filestore/`
(`/var/imq/instances/`*brokerName*`/filestore/` on Solaris)

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

The *brokerName*`/filestore/` directory is created when the broker instance is started for the first time. The procedure for securing this directory depends on the operating system on which the broker is running.

Solaris and Linux. The permissions on the `IMQ_VARHOME/instances/`*brokerName*`/filestore/` directory depend on the umask of the user that started the broker instance. Hence, permission to start a broker instance and to read its persistent files can be restricted by appropriately setting the umask. Alternatively, an administrator (superuser) can secure persistent data by setting the permissions on the `IMQ_VARHOME/instances` directory to 700.

Windows. The permissions on the `IMQ_VARHOME/instances/`*brokerName*`/filestore/` directory can be set using the mechanisms provided by the Windows operating system that you are using. This generally involves opening a properties dialog for the directory.

Plugged-in Persistent Store

A broker using plugged-in persistence writes persistent data to a JDBC-compliant database.

For a database managed by a database server (for example, an Oracle database), it is recommended that you create a user name and password to access the MQ database tables (tables whose names start with "IMQ"). If the database does not allow individual tables to be protected, create a dedicated database to be used only by MQ brokers. See the database vendor for documentation on how to create user name/password access.

The user name and password required to open a database connection by a broker can be provided as broker configuration properties. However it is more secure to provide them as command line options when starting up the broker (see MQ *Administrator's Guide*, Appendix A, "Setting Up Plugged-in Persistence").

For an embedded database that is accessed directly by the broker via the database's JDBC driver (for example, a Cloudscape database), security is usually provided by setting file permissions (as described in "Built-in Persistent Store," above) on the directory where the persistent data will be stored. To ensure that the database is readable and writable by both the broker and the `imqdbmgr` utility, however, both should be run by the same user.

JAR Files for Client Applications

Client applications need to be able to access JNDI JAR files (`jndi.jar`) even if the applications are not directly using JNDI to look up MQ administered objects. This is because JNDI is referenced through the Destination and ConnectionFactory classes.

JNDI JAR files are bundled with JDK 1.4, and therefore jndi.jar does not have to be in the classpath if you are using this JDK version. However, if you are using a version of JDK earlier than 1.4, then `jndi.jar` has to be in your classpath setting (along with `imq.jar` and `jms.jar`).

If you are using JNDI to look up MQ administered objects, you must also include `providerutil.jar` plus either `fscontext.jar` (if you are using the file-system context) or `ldap.jar` (if you are using the LDAP context) in your classpath. If you are using JDK 1.4, you do not need to include `ldap.jar`, since it is bundled with JDK 1.4.

Client Out of Memory Errors

If you are running a client application that deals with large messages or many small messages, it may encounter `OutOfMemoryError` errors. The client runtime does not have a memory leak--it just has insufficient memory to copy the messages off the network and deliver them to your client.

To eliminate these `OutofMemoryError` errors, increase the maximum Java heap size. You can do this by passing the appropriate command line option to the `java` or `jre` command.

On Java2 (formerly code-named "JDK 1.2"), use the `-Xmx` option. For example:

```
java -Xmx128m MyClass
```

Please note these limitations:

- The maximum limit of the VM's memory allocation pool (heap size) depends on both the operating system and the JDK release. Please check the JDK documentation for restrictions.
- The size of the VM's memory allocation pool must be less than or equal to the amount of virtual memory available on the system.

How to Report Problems

To report a problem, send mail to `imq-feedback@sun.com`.

If you have a support contract and you have problems with MQ, contact Sun ONE customer support using one of the following mechanisms:

- Sun ONE online support web site at `http://www.iplanet.com/support/online/`

  From this location, the CaseTracker and CaseView tools are available for logging problems.

- 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

For More Information

Beyond the MQ 3.0 documentation, you can find additional information as indicated below.

Discussion Forums

jmq-interest List

A discussion forum is available for MQ customers. It provides a place for customers to exchange ideas on MQ-related topics and share problem-solving tips and techniques.

To subscribe to the jmq-interest list, send email to `listserv@java.sun.com` and include a message like the following in the message body. Please supply the appropriate data for the first and last name.

```
subscribe jmq-interest firstname lastname
```

To unsubscribe to the list, send email to `listserv@java.sun.com` and include in the message body:

```
signoff jmq-interest
```

NOTE The `jmq-interest` forum is meant for general MQ issues. For specific questions about MQ 3.0, send your questions to `imq-feedback@sun.com`.

Java Technology Forums

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

```
http://forum.java.sun.com
```

Sun ONE Information

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

- MQ website — `http://www.sun.com/software/products/message_queue/home_message_queue.html`
- Release notes and other documentation — `http://docs.iplanet.com/docs/manuals/javamq.html`

  If this URL is discontinued you will be able to access MQ 3.0 documentation from: `http://www.sun.com/software/products/message_queue/home_message_queue.html`
- Support Services — `http://www.sun.com/service/support/software/iplanet/index.html`
- Professional Services information — `http://www.sun.com/service/sunps/iplanet/`
- Developer information — `http://developer.iplanet.com/`
- Learning solutions — `http://www.sun.com/software/training/`
- Product data sheets — `http://www.sun.com/software/html`