Oracle® Java CAPS JMS Reference

Part No: 821–2554
March 2011

120126@25097

Contents

Java CAPS JMS Reference

The following sections provide information on the implementation of JMS in Java CAPS.

Introduction to JMS

This topic provides a basic introduction into how the Java Message Service (JMS) is implemented and used in Java CAPS. It includes the following sections:

JMS and Java CAPS

The following topics provide information about the Java CAPS implementation of JMS:

Java Message Service

The Java Message Service is a Java API used for sending and receiving messages. It is vendor-independent, and is used almost universally in enterprise messaging systems such as that included in Java CAPS. JMS provides a standard, currently embodied in the JMS version 1.1 specification, which is an integral part of the Java Enterprise Edition (Java EE) Platform. The features of the JMS version 1.1 specification have been widely adopted in the Java CAPS JMS implementation, as described in this document. The use of JMS allows loosely coupled, reliable, asynchronous interactions among Java EE components and legacy systems capable of messaging.

Version 1.1 of the JMS API includes the following features:

- Message-driven beans, which enable the asynchronous consumption of JMS messages
- Message send and receive calls that can participate in Java Transaction API transactions
- Java EE Connector Architecture (JCA) interfaces that enable JMS implementations from different vendors to be externally plugged into a Java EE 5 application server

Additionally, the Java EE platform's Enterprise JavaBeans (EJB) container architecture provides the following enhancements the JMS API:

- Concurrent consumption of messages
- Support for distributed transactions, so that database updates, message processing, and connections to EIS systems using the Java EE Connector Architecture can all participate in the same transaction context

JMS Message Servers

JMS message servers provide the global messaging protocols, such as the routing and delivery of messages, and connection to the JMS database.

By default, Oracle Java CAPS Enterprise Service Bus contains the following message server options:

- Sun Java System Message Queue
- Sun JMS IQ Manager

The Sun Java System Message Queue is the current default message server for Java CAPS. You can also connect to various JMS servers from Java CAPS projects.

The Sun JMS IQ Manager was the default JMS message server implementation for eGate Integrator, the predecessor to Oracle Java CAPS Enterprise Service Bus. This software is installed automatically if you select *Complete* when installing from the GUI, or as an option if

you use any other installation method. The JMS IQ Manager conforms to the Java Message specification 1.1 and supports both topic (publish-and-subscribe) and queue (point-to-point) messaging styles. "Inside the JMS IQ Manager" on page 15 includes information about how the JMS IQ Manager processes messages, in concert with the JMS clients. "JMS IQ Manager Runtime Configuration" on page 90 provides detailed information about the JMS IQ Manager property options.

JMS Message Destinations

A message destination is a container for stored data, and can follow either the JMS topic or queue model.

- A *topic* is a message destination that conforms to the publish-and-subscribe messaging paradigm
- A *queue* is a message destination that conforms to the point-to-point messaging paradigm

Each message destination has at least two JMS Clients associated with it: a *producer* client at its input, and a *consumer* client at (each) output. JMS message destinations are discussed in *Developing Java CAPS Projects*.

JMS Clients

JMS clients provide the local messaging protocols, such as message persistence and delivery semantics, for messages being propagated between Project components. Together with the JMS message server, they constitute a *JMS provider*.

JMS clients are of two basic types, *producers* and *consumers*, or a combination of both. If associated with a queue, these types become queue *senders* and *receivers*, respectively. If associated with a topic, they become topic *publishers* and *subscribers*, respectively.

JMS client configuration is discussed in "JMS Client Configuration" on page 81.

JMS Object Type Definitions

The JMS Object Type Definition (OTD) acts as a "wrapper" around a message or connection, allowing Collaborations to read from and write to topics or queues. It indicates to the Collaboration which topic or queue it expects to receive messages from or send messages to, and allows you to build the JMS business rules. The JMS OTD and its properties are discussed in "JMS Object Type Definitions" on page 35. JMS OTD methods are described in "JMS OTD Methods" on page 41 and "JMS Message Methods" on page 65.

JMS Library File

JMS methods are contained in the JMS library file. In Java CAPS 6, this file is located in the following path:

```
...\appserver\domains\domain1\lib\com.stc.jms.stcjms.jar
```

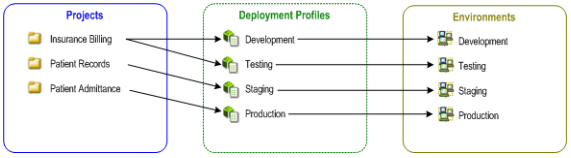
Implementing JMS in Java CAPS Projects

This section includes the following topics and instructions:

Integration Model

The following diagram illustrates the integration model used by Java CAPS.

FIGURE 1 Oracle Java CAPS Enterprise Service Bus Integration Model



This model separates the business logic from the physical system used to perform the logical operations at runtime.

- Each *Project* is a logical construct representing a specific business process.
- Each *Deployment Profile* specifies how Projects are deployed to runtime Environments.
- Each *Environment* is a graphical construct representing a physical system in which Projects can run.

In this model, any of the Projects can be deployed to any of the Environments by means of the mapping defined in the Deployment Profiles. The example in the figure shows that the patient admittance Project is already in the production phase and therefore was deployed using the *production* Deployment Profile. The patient records Project is in the staging phase and was therefore deployed to the staging Environment using the *staging* Deployment Profile. The insurance billing Project is still being developed and tested, and therefore it is deployed to development and testing by means of the *development* and *testing* profiles.

▼ To Implement JMS Following the Java CAPS Model

1 For each Project, specify the message destinations and configure JMS connections for the business process.

2. **For each Environment, assign message servers to Logical Hosts (domains) and configure the message servers.**

3. **For each Project, create a Deployment Profile to associate the message destinations and JMS connections with the message servers.**

Creating and Configuring Message Destinations

In a Java CAPS Project, you specify the business logic for the Oracle Java CAPS Enterprise Service Bus implementation. For each of the Project components you specify logical properties; these properties are independent from the physical implementation. For JMS-related components, Projects are where you add and name message destinations, by dragging and dropping topics and queue icons onto the Connectivity Map canvas. See *Developing Java CAPS Projects* for information.

Creating OTDs and Collaborations

After having added the message destinations, you then create the Object Type Definitions (OTDs) and Collaboration Definitions. For any Java-based Collaboration Definition that reads from or writes to a JMS message destination, you must add the JMS web service. For information on the Java methods used in these Collaboration Definitions, see "JMS OTD Methods" on page 41 and "JMS Message Methods" on page 65.

Configuring JMS Clients

Links between message destinations and their subscribers and publishers, display a JMS Client icon. By double-clicking the icon in the Connectivity Map, you can configure local connection properties such as persistent or non-persistent delivery mode, XA, and concurrent processing. Details of the configuration properties for JMS Clients are given in "JMS Client Configuration" on page 81.

Configuring Message Servers

In the runtime Environment, you specify which message servers are used, and which Logical Hosts (domains) they are to run on. Once you add a message server to a domain, you specify the physical configurations for the message server. You can configure global properties for JMS messaging such as the port number, message delivery order, tuning configurations, journaling options, and diagnostic options. Procedures for defining your Environment are described in *Creating a Runtime Environment*. Details of the configuration properties for the JMS IQ Manager are given in "JMS IQ Manager Runtime Configuration" on page 90.

Creating Component Mappings

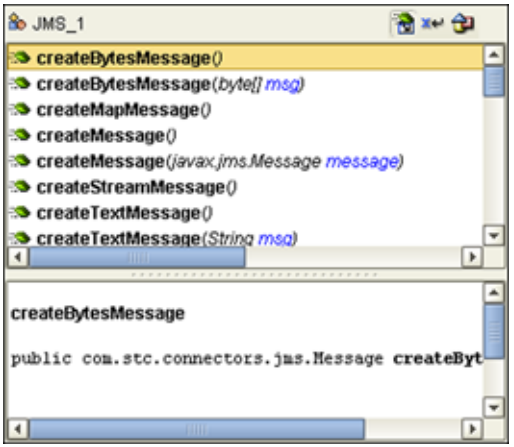
When you define a Deployment Profile, you create mappings between Projects and Environments. In the Deployment Profile, you specify which components of the business

process are located on which systems in a specific Environment. For the JMS, you specify which message destinations run on a particular message server. Note that inbound and outbound message destinations must be deployed to the same server. For more details, see *Deploying Java CAPS Projects*.

Using the JMS OTD in Collaboration Definitions

To enable Java based Collaboration Definitions to read from and write to topics or queues, you must add the JMS OTD to the Collaboration Definition. The JMS OTD contains a set of Java methods that are used by the JMS client to manage the JMS session. These methods are accessed by displaying the context menu for the OTD root node, and then choosing `Select method to call` from the menu. This option displays the method browser, shown in the following figure. You can also drag the node to the mapping area to display the browser.

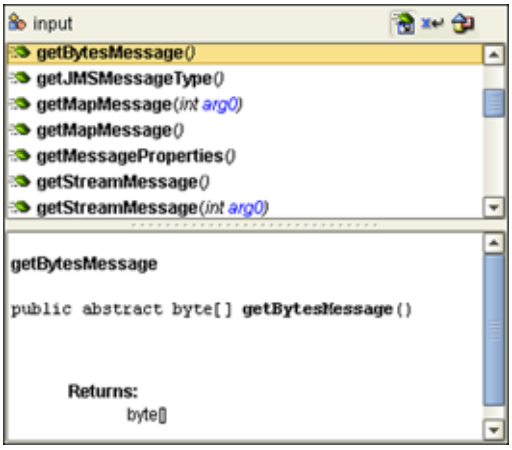
FIGURE 2 Java Method Browser



Using JMS Messages in Collaboration Definitions

To create Java based Collaboration Definitions that implement an existing web service, you must add the JMS `receive`, `receiveWait`, or `send` web service operation to the Collaboration Definition. Each of these operations has a set of Java methods that is available for use with it. These methods are accessed by displaying the context menu for the `input` or `output` node, and then choosing `Select method to call` from the menu. This option displays the method browser, shown in the following figure. You can also drag the node to the mapping area to display the browser.

FIGURE 3 Java Method Browser



Inside the JMS IQ Manager

This topic describes the functional behavior of the JMS IQ Manager and its interaction with JMS clients. It addresses topics such as the JMS IQ Manager database, message processing, performance, and optimization. The following links lead to detailed descriptions of the listed subtopics.

JMS Messaging Features

This topic provides a brief overview of messaging features offered by the JMS IQ Manager working together with the JMS clients.

Message Delivery Order

The JMS IQ Manager provides the following special facilities to maintain message order in concurrent processing and across message destinations. These facilities are in addition to those mandated by the Java Message Server specification.

- Configuring the JMS IQ Manager for special first-in, first-out (FIFO) ordering modes for queues
- Specifying a set of message destinations (a time order group) for which fully serialized processing occurs
- Configuring topics and queues for concurrent or serial processing

The following sections describe each method of processing order. For detailed information about processing order, refer to "Message Processing Order" on page 20.

Message Producer Priorities

Oracle Java CAPS Enterprise Service Bus enables you to set message priorities for topic publishers and queue senders. The priority level causes all messages produced by the client to have that same priority level. For more information, refer to "Message Producer Priorities" on page 25.

Distributed Transactions

The JMS IQ Manager enables you to configure JMS Client properties for distributed transactions using the XA protocol. For more information, refer to "Transaction Mode" on page 89.

Security

The Oracle Java CAPS Enterprise Service Bus provides role-based security for the JMS IQ Manager by requiring authentication by means of SSL or LDAP. When authentication is enabled, access to the JMS IQ Manager is only granted when the connection has a valid user ID and password.

JMS IQ Manager security is enabled by default. To disable security, refer to *Using LDAP with Java CAPS*.

Runtime Management

Oracle Java CAPS Enterprise Service Bus provides two alternatives for managing JMS IQ Managers:

- **Enterprise Manager**

  Enterprise Manager offers a graphical interface containing comprehensive runtime management functions. You can monitor message destinations, and view message properties and payloads. For more information, see *Using Enterprise Manager Management Application in Java CAPS*.

- **MS Control utility**

  The MS Control utility is a command-line utility that enables you to manage many advanced aspects of the JMS IQ Managers. For details, refer to "JMS Provider Management" on page 107.

Note – Other message servers supported by Java CAPS have their own runtime management tools.

JMS IQ Manager Database

The JMS IQ Manager uses the JMS IQ Manager database to store persistent messages. The database is also used to store messages that are larger than can be kept in the JMS IQ Manager memory, which is determined by the cache size setting (by default 0.5 MByte for Windows and 1 MByte for UNIX). By default, JMS clients are configured for persistent messaging; therefore, in a default configuration, the database is used to store these messages. The messages are stored until they are consumed or until the expiration period expires, which is 30 days by default. See "Maximum Lifetime" on page 97.

Database Files

The database resides in the message server folder on the Logical Host. The database consists of a number of database files called *segments*. A segment is a disk-space store that is memory-mapped on the server. The segments act together to form the equivalent of a sequential database. By default, these files are named `stcms*.dbs`.

The JMS IQ Manager creates four segments in the database when it starts up initially. The default size of a segment is 8 MBytes on Windows and 16 MBytes on UNIX. The JMS IQ Manager creates as many segments as necessary. Before running your Java CAPS Project, it is important to set the segment size to a larger value than the largest transaction the JMS IQ Manager may need to process, since the JMS IQ Manager cannot accommodate a transaction that is larger than the segment size.

Note – The transaction size is the sum of the sizes of the individual messages involved.

You can configure such properties as database filenames, segment size, and the maximum and minimum number of segments created. For more information, refer to "Segment Properties" on page 92.

Database Location

The JMS IQ Manager database resides in the directory specified by the JMS IQ Manager `Data directory` property as described in "Data Directory" on page 93. The standard location for the database files is described in "Database Configuration and Operation" on page 18

If journaling is enabled, the instance1 directory also contains a `Journal` directory, unless another location has been specified for the `Journal Directory` property. The Journaling directory holds the journaling database files. For information, refer to "Journal Directory" on page 98. Journaling is *disabled* by default.

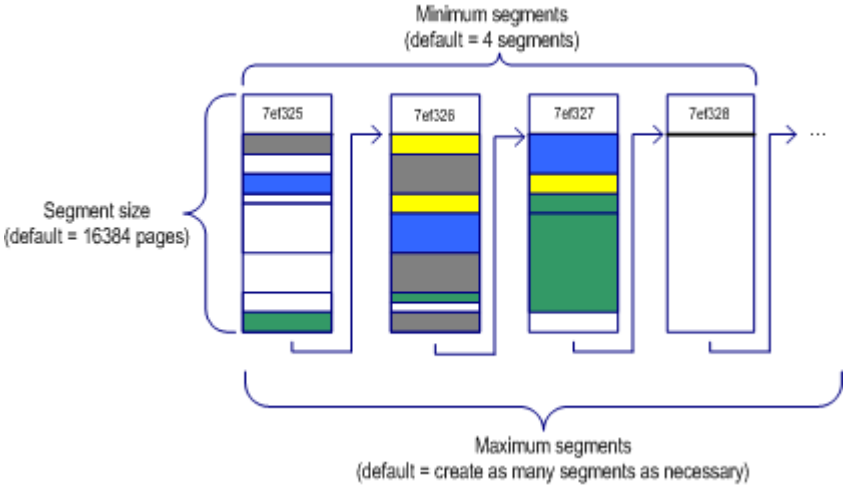
Database Configuration and Operation

The default configuration for the JMS IQ Manager database is as follows:

- The database resides in the following directory:`..\logicalhost\is\domains\`*domain_name*`\stcms\instance_1`
- The segment filenames are `stcms*.dbs`.
- The number of segments created initially for the database is `4`.
- The size of each segment is `8` MBytes on Windows and `16` MBytes on UNIX.
- There is no limit to the number of segments that can be created.

When the JMS IQ Manager starts up, the database consists of four segments; the total size of the database is `32` MBytes on Windows and `64` MBytes on UNIX.

FIGURE 4 JMS IQ Manager Database Structure



On startup, the JMS IQ Manager performs the following operations:

1. It allocates sufficient disk space to hold the minimum number of segments.

   The preceding figure shows a JMS IQ Manager allocation of four segments, numbered `7ef325` through `7ef328`.

2. As messages arrive, they are appended to the first segment until the segment is full.

   When a segment is full, the JMS IQ Manager stores subsequent messages in the first free segment.

   The preceding figure shows that the third segment, `7ef327` in file `stcms7ef327.dbs`, is almost full.

3. If no segment is free, the JMS IQ Manager allocates a new segment if possible.
4. Memory is released as soon as the consumer has acknowledged the message or committed the transaction.
5. When all messages in a segment have expired or been removed from their queues, the JMS IQ Manager cleans up the segment, freeing it for reuse.

   In the preceding figure , the first segment (`7ef325`) has several segments that are white, indicating the slot is marked eligible. The segment is therefore almost ready for cleanup.

Message Processing Order

You have several options for controlling the message processing order in Java CAPS Projects:

- You can specify first-in, first out (FIFO) ordering modes, or a set of message destinations with a specific processing order, when you configure the JMS IQ Manager. See "JMS IQ Manager Delivery Modes" on page 20.
- You can specify message processing, either connection consumer or serial mode, at the JMS client level. See "JMS Client Delivery Modes" on page 23.

JMS IQ Manager Delivery Modes

For a single consumer with a single process, processing for queues is fully serialized, by default. The process of processing a message is as follows:

1. The receiver requests, or is ready to receive, a message
2. The receiver receives the message
3. The receiver processes the message

When multiple receivers or multiple processes within a single receiver subscribe to the same message destination (queues only), you have a choice of three first-in, first-out (FIFO) delivery modes, as listed below. These ordering modes apply globally to all queues in the Java CAPS Project

- **Fully concurrent**

  Receivers can retrieve messages when all older messages have been received, or are being received, and can commit messages in any order (without using time sequence).
- **Protected concurrent**

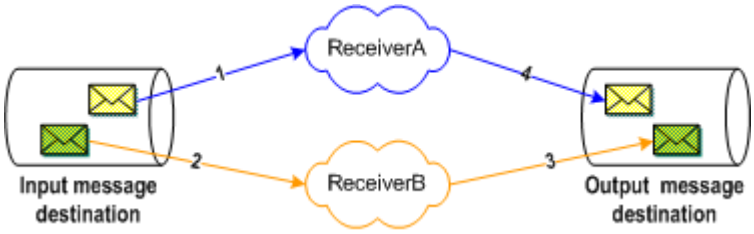
  Receivers can retrieve messages when all older messages have been received or are being received, but must commit using time sequence.
- **Fully serialized**

  Receivers read a messages only after all messages have been received and commit messages using a time sequence.

Fully Concurrent Processing

In fully concurrent mode, receivers can retrieve messages from a destination only when all older messages have been received or are in the process of being received. Receivers can then commit messages without restrictions. By default, JMS IQ Managers use fully concurrent processing for queues.

FIGURE 5 Fully Concurrent Processing



The figure above shows a sample delivery sequence for fully concurrent processing. In steps 1 and 2, the receivers retrieve their messages from the input queue. Both receivers must wait until each consumer has retrieved its messages, or is in the process of retrieving them, before they are able to commit messages to the output destination.

As steps 3 and 4 infer, receivers can commit messages in any order; therefore, messages can be committed out of sequence, which might not be acceptable. For example, a patient release record may be committed before the patient admittance record is committed. The following table indicates the benefits and drawbacks of fully concurrent processing.

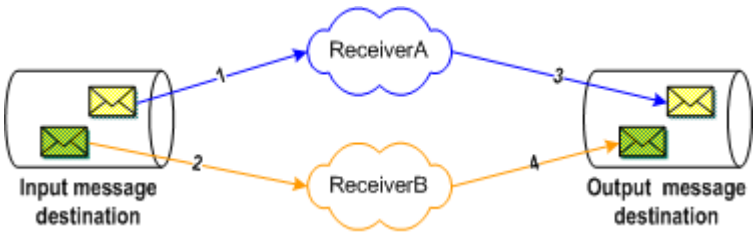
TABLE 1 Tradeoffs for Fully Concurrent Processing

Benefits	Drawbacks
Provides the highest performance.	Delivery is not time-sequenced.
Receivers are not hampered by other receivers.	

Protected Concurrent Processing

In protected concurrent mode, receivers retrieve messages just as in fully concurrent mode, after all messages have been received or are being received. Messages can only be committed, however, if all older messages have previously been committed.

FIGURE 6 Protected Concurrent Processing



The figure above shows a sample delivery sequence for protected concurrent processing. In steps 1 and 2, the receivers retrieve their messages from the input queue. Both receivers must wait until each consumer has retrieved, or is in the process of retrieving, its messages before being able to commit messages to the output destination. ReceiverB might be ready to commit its message before ReceiverA, but must wait until ReceiverA commits its message (step 3). Only when ReceiverA's message has been committed, can ReceiverB commit its message (step 4).

Protected concurrent processing thus is a more workable solution in a scenario where a Project deals with messages such as patient records, where the admittance record must be committed before the release record.

The following table shows the benefits and drawbacks of protected concurrent processing.

TABLE 2 Tradeoffs for Protected Concurrent Processing

Benefits	Drawbacks
Provides better performance than serialized processing. Messages are delivered by time sequence.	Provides lower performance than fully concurrent processing.

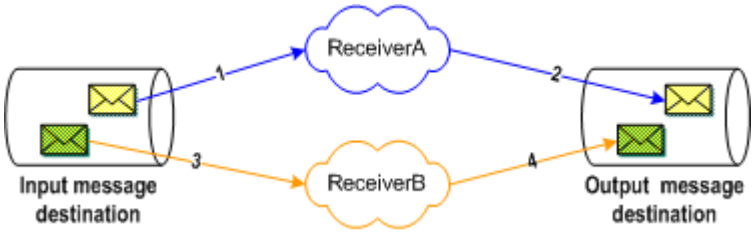
You specify protected concurrent processing for JMS IQ Managers with the `Protected Concurrent Queues` property as described in "Special FIFO Mode Properties" on page 100.

Note – By design, protected concurrent FIFO works only within one transaction, either in the form of one session or in the form of one XA transaction on the same server. You should always use XA when using the protected concurrent FIFO mode with a message-driven bean (MDB), since Java CAPS does not support single sessions with MDBs.

Fully Serialized Processing

In fully serialized mode, receivers can only retrieve messages after all older messages have been received *and* committed.

FIGURE 7 Fully Serialized Processing



The figure above shows a sample delivery sequence for serialized processing. In step 1, ReceiverA retrieves its message. ReceiverB might at this point be ready to receive its message, but must wait until ReceiverA has committed its message. After ReceiverA commits the message in step 2, ReceiverB can then retrieve and commit its message (steps 3 and 4).

The following table shows the benefits and drawbacks of protected concurrent processing.

TABLE 3 Tradeoffs for Fully Serialized Processing

Benefits	Drawbacks
Guaranteed delivery by time sequence.	Provides the lowest performance of all FIFO modes.

You specify fully serialized processing for JMS IQ Managers with the `Fully Serialized Queues` property as described in "Special FIFO Mode Properties" on page 100.

Serial Processing Across a Destination Group

You can also specify delivery order specifically for a set of topics and queues (time order groups). For these groups, consumers can only receive messages when all other messages in the time order group have been received or are in the process of being received. For information on specifying a time order group, refer to "Time Dependency Properties" on page 102.

JMS Client Delivery Modes

The delivery order options above are configured for the JMS IQ Manager. The Oracle Java CAPS Enterprise Service Bus JMS implementation enables you to configure topic subscribers as connection consumers to improve message throughput through concurrent processing. You can set the JMS client configuration with the `Concurrency` property as described in "Concurrency" on page 83.

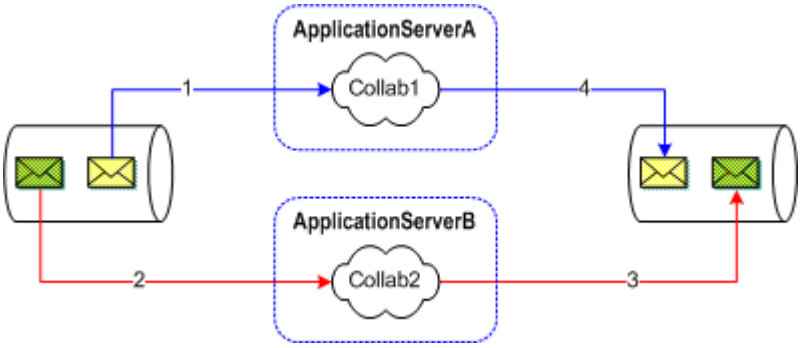
The use of connection consumers increases message processing performance by enabling concurrent processing using multiple threads. You can specify the number of message driven beans (MDBs) or the server session pool to assign to a JMS Collaboration to process messages concurrently. When you use connection consumer with fully concurrent or protected

concurrent FIFO processing, this default setting allows the integration server to assign multiple threads to execute the Collaboration on a particular message destination.

For queues, you can also process messages concurrently using multiple integration servers; these integration servers may run on different systems.

Using a JMS client connection consumer affects the message processing order. For example, consider the scenario shown in the following figure. The JMS IQ Manager is set to fully concurrent FIFO processing. However, each Collaboration on each integration server retrieves messages as they come in, and is able to commit them unrestricted to the queue. Therefore, although the JMS IQ Manager is configured for fully concurrent FIFO processing, message order cannot be guaranteed.

FIGURE 8 Multiple Application Server Configuration



The concurrency mode affects FIFO processing in several ways. The following table lists how the JMS client concurrency mode settings affect the JMS IQ Manager FIFO mode selections for topics. For topics, only one integration server per subscriber can be used.

TABLE 4 Concurrency/FIFO Mode Interaction: Topics

Concurrency Mode	Fully Concurrent FIFO Mode
Serial mode	Fully serialized
Connection consumer	No strict order maintained

The following table lists how the JMS client concurrency mode settings affect the JMS IQ Manager FIFO mode selections for queues.

TABLE 5 Concurrency/FIFO Mode Interaction: Queues

		FIFO Mode		
Application Servers	**Concurrency Mode**	**Fully Serialized**	**Protected Concurrent**	**Fully Concurrent**
Single	Serial	Fully serialized	Protected concurrent	Fully serialized
		No concurrency	No concurrency	No concurrency
	Connection consumer	Fully serialized	Protected concurrent	No strict order
		No concurrency	Concurrent	Concurrent
Multiple	Serial	Fully serialized	Protected concurrent	No strict order
		No concurrency	Concurrent	Concurrent
	Connection consumer	Fully serialized	Protected concurrent	No strict order
		No concurrency	Concurrent	Concurrent

Message Producer Priorities

Oracle Java CAPS Enterprise Service Bus enables you to set message priorities for topic publishers and queue senders. The priority is specified at the JMS client level; therefore, the level you specify causes all messages produced by that client to have that same priority level, unless overridden for a specific Collaboration. For example, if you set the priority level to 2, all messages sent by that client have message priority level 2. The default message priority is 4.

The Oracle Java CAPS Enterprise Service Bus message priority implementation adheres to the recommended standards in the Java Specification: in most circumstances, messages with higher priorities are delivered before message with lower priorities.

For information about setting the priority level for topic publishers and queue senders, refer to "Priority" on page 89.

You can also specify message priorities in specific Collaboration Definitions with the Collaboration Definition Editor. Collaboration message priorities override message priorities specified at the JMS client level.

Message Redelivery and Redirection

JMS *receive* methods provide for the redelivery of messages that are rolled back when the message cannot be delivered, as in the figure below. JMSJCA, an implementation of the Java Connector Architecture 1.5, provides additional flexibility in the way messages are redelivered, and adds the option of redirecting the messages to another destination. The JMS IQ Manager currently supports these JMSJCA features.

FIGURE 9 Message Rollback and Redelivery

The reason for rolling back a message can be either permanent or transient. If it is transient, the message will eventually be delivered after some number of retries; but if the number of retry/rollback cycles required to redeliver the message becomes large, system resources can be negatively affected. To avoid this situation, a series of progressive delays following a prescribed number of failed retries is introduced into the redelivery process. The default behavior is described in the following table.

Total Number of Failed retries	Delay (milliseconds)
3	25
5	50
10	100
20	1000
50	5000

You can override this default behavior by configuring your own custom characteristics. You specify the actions you want to be taken after a message has been rolled back by means of a specially formatted string. You append this string to the message server URL when you configure the JMS IQ Manager. The formatting of this string is described in the following sections.

Redelivery Options

You can use any of the following redelivery options:

Progressive Delay

The format of such a delay is *retries:delay*, where the number of retries is counted from the original rollback and the delay time is given in milliseconds. The maximum allowed delay is five seconds (5000 ms). The following example shows a string that specifies a delay of 1000 milliseconds following five failed retries. In other words, it specifies no delay for the first five attempts at redelivery, then a one-second delay for each subsequent attempt.

Example 1

```
5:1000
```

You can also cascade these delay actions to become progressively longer as the number of retries increases. Entries are separated by a semicolon followed by a space. The following example shows a string that specifies a one-second delay following five failed attempts at redelivery, ana a five second delay following a total of ten failed attempts.

Example 2

```
5:1000; 10:5000
```

Delay and Redirect

After a certain number of failed redelivery attempts, you may want to redirect the message to a different target destination, such as a dead-letter queue. The format for redirecting is *retries*`:move(args)`, where the arguments can be `queue:`*target*, `topic:`*target*, or `same:`*target*.

The argument component `same` specifies the same kind of message destination as the message source. That is, if the message was received from a queue, it will be sent to a queue; if the message was received from a topic, it will be sent to a topic.

The argument component *target* can be any string and can include the character `$`, which is automatically replaced with the original destination name.

The following example shows a string that specifies a one-second delay following five failed attempts at redelivery, a 5–second delay following a total of ten failed attempts, then redirects the message to a dead-letter queue named `mydlq` after a total of 50 failed attempts.

Example 3

```
5:1000; 10:5000; 50:move(queue:mydlq)
```

The following example shows a string that specifies a one-second delay following five failed attempts at redelivery, a five second delay following a total of ten failed attempts, then redirects the message to a dead-letter queue after a total of 50 failed attempts. If the message was received from a source destination named `Queue1`, the message will be redirected to a target destination named `dlqQueue1error`.

Example 4

```
5:1000; 10:5000; 50:move(queue:dlq$error)
```

Delay and Delete

After a certain number of failed redelivery attempts, you may want to simply delete the message. The format for deleting is *retries:delete*. The following example shows a string that specifies a one-second delay following five failed attempts at redelivery, a five-second delay following a total of ten failed attempts, then deletes the message.

Example 5

```
5:1000; 10:5000; 50:delete
```

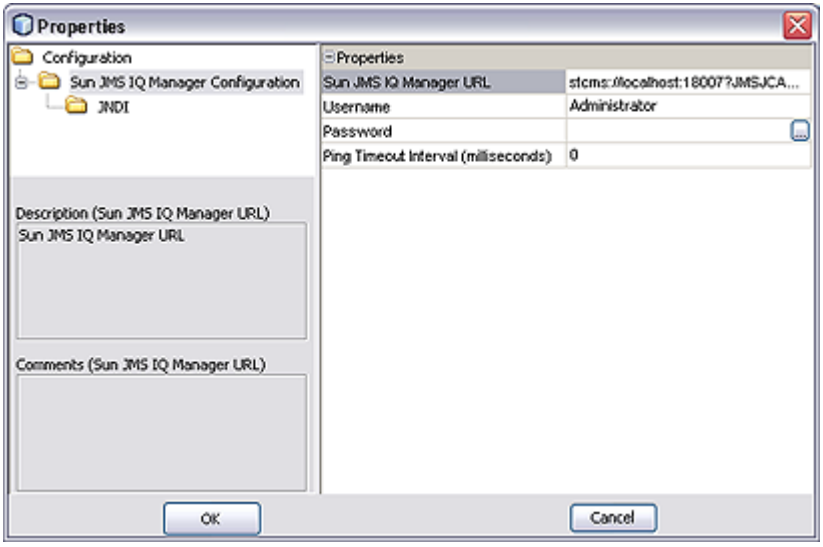
Specifying Redelivery Options in the JMS IQ Manager

You can specify the actions you want to be taken after a message has been rolled back by appending a redelivery-handling string to the message server URL when you configure the JMS IQ Manager. These actions then override the default actions for all JMS clients interacting with the JMS IQ Manager. The format for this string is:

`?JMSJCA.redeliveryhandling=`*action*

where *action* is the string specifying the delay/redirection/deletion as given in the examples of the preceding section. The following figure illustrates how this string would be specified in the property dialog in the NetBeans IDE.

FIGURE 10 JMS IQ Manager Configuration Properties - Redelivery Example

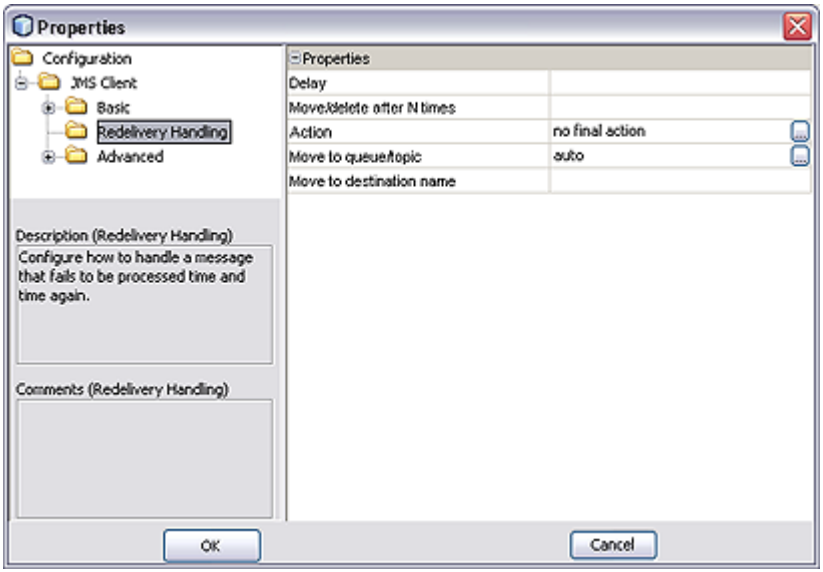


Specifying Redelivery Options in a JMS Client

You can also specify the actions you want taken for a specific JMS client when specifying the configuration properties for that client, as shown in the following figure. If you do so, the properties you specify for the JMS client will override the redelivery properties specified by the JMS IQ Manager for that client only.

For descriptions of the JMS client redelivery configuration properties, see "JMS Client Configuration" on page 81, or *Configuring JMS Clients*.

FIGURE 11 JMS Client Configuration Properties - Redelivery Handling



Enqueued Message Properties

Messages contained within a message destination (queue or topic) are assigned certain properties for the length of time that they reside in the message destination. These properties are displayed when you check the status of the message destination using either Enterprise Manager or the command-line MS Control utility. See "JMS Provider Management" on page 107.

Note – Whenever unconsumed messages exist in the topic or queue, the message having the *first enqueue time* will have the *minimum sequence number* and the message having the *last enqueue time* will have the *maximum sequence number*.

Enqueue Time

Each message is given a message enqueue time by the JMS IQ Manager when a message is added into a queue or topic. This value is determined by the server side and is unique across a topic or a queue. For a transaction session, the message enqueue time is determined by the JMS IQ Manager when a message is committed. For a non-transaction session, the message enqueue time is determined by the JMS IQ Manager when the JMS IQ Manager receives the message.

The *first enqueue time* of a topic or queue is the enqueue time of the first unconsumed message in the topic or queue. The *last enqueue time* of a topic or queue is the enqueue time of the last

unconsumed message in the topic or queue. If no unconsumed message exists in the topic or queue, then the last enqueue time and the first enqueue time are identical, and represent the enqueue time of the most recently consumed message in the topic or queue.

The following example displays the current status of the queue `PTP`:

```
stcmsctrlutil -host localhost -port 24055 -queuestat PTP
Queue Name: PTP
First enqueue time: 03212005:08:33:09
Last enqueue time: 03212005:08:33:10
Number of current receivers: 2
Message count: 4
Messages sent and committed: 245
Min sequence number: 245
Max sequence number: 248
Suspended: No
```

In this example, four messages are in the queue `PTP`. The *first enqueue time* is the enqueue time of the first message, which is `03212005:08:33:09`, and the *last enqueue time* is the enqueue time of the fourth message, which is `03212005:08:33:10`. If all four messages are consumed, both the first and last enqueue times become the enqueue time of the fourth message, which is `03212005:08:33:10`.

In the initial state, where the JMS IQ Manager has not received or consumed any messages in the topic or queue, both the first and last enqueue times are `N/A`.

Note – You should not change the clock of the computer on which the JMS IQ Manager is running once the JMS IQ Manager is configured. If you do, the message enqueue time will be incorrect.

Sequence Number

Each message has a message sequence number that is assigned by the JMS IQ Manager when the message is added to the queue or topic. The message sequence number provides a message index, represented as a long integer value, and is unique across a topic or queue. For a transaction session, the message sequence number is determined by the JMS IQ Manager when a message is committed. For a non-transaction session, the message sequence number is determined by the JMS IQ Manager when the JMS IQ Manager receives the message.

The *minimum sequence number* of a topic or queue is the sequence number of the first unconsumed message in the topic or queue. The *maximum sequence number* of a topic or a queue is the sequence number of the last unconsumed message in the topic or queue. Because all messages in the topic or the queue are sorted and indexed by sequence number, the first unconsumed message has the minimum sequence number, and the last unconsumed message has the maximum sequence number, among those residing in the topic or queue. If no unconsumed message exists in the topic or queue, the minimum sequence number and the

maximum sequence number are identical and represent the next available sequence number. This number will be assigned to the next message received by the topic or queue.

The following example displays the current status of the queue `PTP`:

```
stcmsctrlutil -host localhost -port 24055 -queuestat PTP
Queue Name: PTP
First enqueue time: 03212005:08:33:09
Last enqueue time: 03212005:08:33:10
Number of current receivers: 2
Message count: 4
Messages sent and committed: 245
Min sequence number: 245
Max sequence number: 248
Suspended: No
```

In this example, four messages are in the queue `PTP`. The `minimum sequence number` is the sequence number of the first message, which is `245`, and the `maximum sequence number` is the sequence number of the fourth message, which is `248`. If all four currently resident messages are consumed, both the minimum and maximum sequence numbers will represent the next available sequence number, which is `249`. This number will be assigned to the next incoming message.

In the initial state, where the JMS IQ Manager has not received or consumed any messages in the topic or queue, both the minimum and maximum sequence numbers are `0` (zero).

Performance Issues

The large assortment of configuration parameters permits a high degree of control over processing speed, memory use, and disk space. The JMS IQ Manager properties work together to enable you to fine-tune your system according to load and hardware constraints. For information, refer to "Segment Properties" on page 92.

Because every message is written to disk, file input/output (I/O) is usually the hardware factor with the largest performance impact. For a disk with adequate I/O speed, highest performance is achieved by holding all messages in server memory continuously until the corresponding segment is cleaned up.

Available server memory can easily be exceeded for systems handling very large messages, so several configuration parameters are provided to help you manage a memory-bound server. See "Throttling Producers" on page 33 for additional information.

To maximize performance:

- Use the fastest disk possible.
- Keep in mind that allocating a new segment requires more time than freeing a cleaned-up segment.

- Set the segment size to lower values, because smaller segments turn over more rapidly and thus provide more effective use of server memory. However, because cleaning up two small segments requires more time than cleaning up one large segment, you can use very large segments to increase performance on systems that are constrained by disk I/O speed rather than memory or space.

Throttling Producers

In addition to disk and memory-management features provided by the operating system, special configuration properties within the JMS IQ Manager specifically deal with messages, message destinations, and producers.

When the amount of JMS IQ Manager memory allocated to messages reaches a certain limit, you can instruct the JMS IQ Manager to stop reading all messages from one or more producers until certain criteria are met. This process is referred to as *throttling* the producer.

Producer throttling is done on a per-message destination basis. This process addresses the two most common reasons for approaching the JMS IQ Manager memory limit in an otherwise well-tuned system:

- **A particular message destination has a period of abnormally heavy traffic.**

  Throttling all producers that feed the message destination gives the destination's consumer a chance to catch up while maintaining normal throughput for other message destinations.
- **A particular consumer fails, causing a backup of all message destinations to which it subscribes.**

  If the problem is transient, then throttling all its producers gives the consumer a chance to catch up on the backlog. If the problem is permanent, then throttling the producers allows unaffected message destinations to flow freely. Meanwhile, the problem can be diagnosed and repaired without taking the server offline.

Three configuration properties govern producer throttling:

- The `Server Throttling Threshold` property sets the *message server* limit. When the JMS IQ Manager is below this threshold, it does not throttle any producers, whether or not they are feeding a destination that has exceeded the `Per-Destination Throttling Threshold`.
- When the `Server Throttling Threshold` has been exceeded and producer throttling is in effect, the `Per-Destination Throttling Threshold` property specifies the *per-destination* limit. The JMS IQ Manager stops reading messages from producers feeding any message destination that has exceeded this limit.
- `Throttling lag` determines how many messages for this topic must be removed from the queue before throttling can stop.

Once throttling has been initiated, the JMS IQ Manager resumes reading messages for the affected message destinations only when one of the following criteria is met:

- The JMS IQ Manager falls below the `Server Throttling Threshold` limit.
- The number of undelivered messages in the affected destination has dropped to less than the difference between the value specified by the `Per-Destination Throttling Threshold` and the value specified by the `Throttling Lag`.

Note – Each message in a message destination counts against the message destination's `Per-Destination Throttling Threshold` limit until the message is dequeued. In particular, a non-transactional message is counted until it has been delivered to all its subscribers, whereas a transactional or XA-compliant message is counted until it has been committed by all consumers.

Example of Producer Throttling and Unthrottling

The following table illustrates a scenario where a JMS IQ Manager exceeds its specified threshold and begins to throttle selected producers. In this example, the JMS IQ Manager uses the following (default) values for throttling properties:

- `Server Throttling Threshold` = 100,000
- `Per-Destination Throttling Threshold` = 1,000
- `Throttling lag` = 100

Two minutes after the server exceeds its limit, Topic_A, which has two subscribers and one publisher, is affected. Its publisher is throttled until the number of undelivered messages can drop below 900. Later, because the JMS IQ Manager is no longer loaded, the same topic builds up an even greater backlog without having its publisher throttled.

TABLE 6 Publisher Throttling Example

Time	Total Messages on All Topics (Server)	The Highest Sequence Number (Messages in Topic_A only)			Comment
		Read from Pub1:	Sent to Sub1:	Sent to Sub2:	
11:37	98604	500	200	75	Server is not yet over limit.
11:38	100307	800	500	150	Server is now over limit, but Topic_A is unaffected because its subscribers are keeping up well enough.
11:39	101283	1100	800	225	Server is still over limit, Topic_A is still unaffected because only 875 messages are undelivered.

TABLE 6 Publisher Throttling Example *(Continued)*

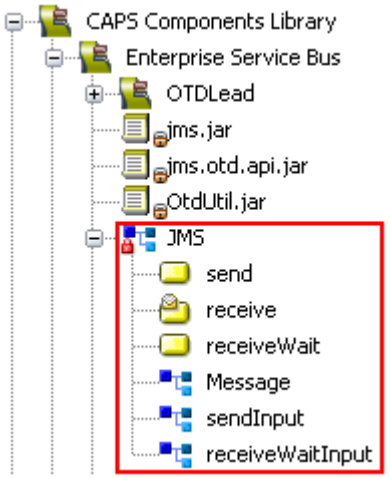
Time	Total Messages on All Topics (Server)	The Highest Sequence Number (Messages in Topic_A only)			Comment
		Read from Pub1:	Sent to Sub1:	Sent to Sub2:	
11:40	103429	1350	1050	300	Now that it has 1050 undelivered messages, Topic_A has crossed the per-destination limit. While the server remains over limit, Pub1 will remain throttled until the number of undelivered messages falls below 900, which is the difference between the value specified by the `Per-Destination Throttling Threshold` and the value specified by the `Throttling Lag`).
11:41	104031	1350	1300	375	Pub1 is throttled, Sub1 is nearly caught up, Sub2 is catching up but still has 975 undelivered messages.
11:42	103204	1350	1350	449	Pub1 is throttled, Sub1 has caught up, Sub2 has 901 undelivered messages, which is still too many.
11:43	102762	1350	1350	451	Although the server is still over limit, it unthrottles Pub1 because the undelivered message count for Topic_A has fallen below 900.
11:44	101095	1375	1370	525	Server is over limit, but Topic_A is unaffected because it has only 850 undelivered messages.
11:45	100028	1575	1500	600	Server is over limit, but Topic_A is unaffected because it has only 975 undelivered messages.
11:46	99248	1900	1700	675	Server is no longer over limit. No publishers are throttled even though Sub2 has more than 1000 undelivered messages.

JMS Object Type Definitions

The JMS Object Type Definition (OTD) is a special type of OTD that allows Collaborations to read from and write to topics or queues. It supplies operators and Java methods for creating, sending, and receiving JMS messages (see "JMS OTD Methods" on page 41).

The JMS OTD is included with Oracle Java CAPS Enterprise Service Bus, and is installed automatically. The template resides in the `CAPS Components Library\Enterprise Service Bus` folder under the NetBeans `Projects` tab, as shown in the following figure.

FIGURE 12 Project Explorer - JMS OTD



The `send`, `receive`, and `receiveWait` nodes represent *web service operations* that are available in the Collaboration Definition Wizard (Java) when you create a Java based Collaboration Definition implementing an existing web service. The `Message`, `sendInput`, and `receiveWaitInput` nodes represent *messages* used with these operations, as listed in the following table.

Message Node	Function
Message	Message for `receive` operation
sendInput	Message for `send` operation
receiveWaitInput	Message for `receiveWait` operation

When you select one of the web service operations for a Java based Collaboration Definition implementing an *existing* web service, the corresponding message becomes available in the Collaboration Definition Editor as the `input` argument. When you create a Java based Collaboration Definition implementing a *new* web service, the three messages become available as both `input` and `output` arguments.

The `send`, `receive`, and `receiveWait` nodes can also be dragged and dropped into the Oracle Java CAPS Business Process Editor as *activities*. The corresponding messages are then displayed in the mapper while performing assigns. See *Designing Oracle Java CAPS Business Process Manager Projects* for additional information.

Message Types

The JMS OTD currently supports the following message types:

- **Message**

  A *Message* carries no payload, and is generally used for event notification. Java methods specifically available for use with this message type are:

  - "createMessage()" on page 44
  - "createMessage(msg)" on page 44

  BytesMessage

  A *BytesMessage* carries a byte array as its payload, and is often used in cases where JMS is simply used as a transport between systems. Java methods specifically available for use with this message type are:

  - "createBytesMessage()" on page 42
  - "createBytesMessage(msg)" on page 43
  - "getBytesMessage()" on page 67
  - "retrieveBytesFromMessage()" on page 72
  - "retrieveBytesFromMessage(arg0)" on page 73
  - "setBytesMessage(arg0)" on page 77

  MapMessage

  A *MapMessage* carries a set of name-value pairs as its payload, and is often used for delivering keyed data. Java methods specifically available for use with this message type are:

  - "createMapMessage()" on page 43
  - "countMapMessage()" on page 65
  - "getMapMessage()" on page 68
  - "getMapMessage(arg0)" on page 68
  - "retrieveMapMessage(arg0)" on page 73
  - "retrieveMapMessageList()" on page 74
  - "storeMapMessage(arg0, arg1)" on page 79

  StreamMessage

  A *StreamMessage* carries a stream of primitive Java types (such as `char`, `double`, and `int`) as its payload, and is often used when delivering primitive application data in a fixed order. Java methods specifically available for use with this message type are:

  - "createStreamMessage()" on page 45
  - "countStreamMessage()" on page 66
  - "getStreamMessage()" on page 70
  - "setStreamMessage(arg0, arg1)" on page 78

  TextMessage

A *TextMessage* carries a string, of type `java.lang.String`as its payload, and is used for exchanging both text messages and XML documents. As such, it is the most often used message type. Java methods specifically available for use with this message type are:

- "createTextMessage()" on page 45
- "createTextMessage(msg)" on page 46
- "getTextMessage()" on page 71
- "retrieveStringFromMessage(arg0)" on page 75
- "retrieveStringFromMessage()" on page 75
- "setTextMessage(arg0)" on page 78

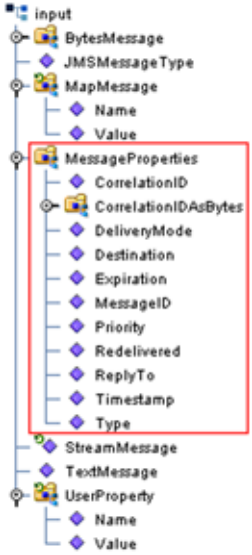
JMS Message Properties

You can set specific message properties in the JMS OTD using the property nodes that are exposed by expanding the appropriate nodes in the Collaboration Definition Editor.

JMS Message Header Properties

You can set the message header properties for both inbound or outbound JMS messages. These property nodes are shown in the following figure, as they appear in the user interface. These same properties are also available for the *output* node in the `receiveWait` operation. When you set these properties in a Java based Collaboration Definition, they are used only by the Collaboration that uses that specific Collaboration Definition.

FIGURE 13 JMS Message Property Nodes (*receive* operation)



The following table shows the allowed values for the properties and the JMS methods used when you set a property in a Collaboration Definition. The methods that are exposed for your use are cross-referenced to the appropriate description. The methods that are not cross-referenced are automatically assigned, and not exposed.

Property	Values	Equivalent JMS Methods
`CorrelationID`	Correlation ID	`getJMSCorrelationID()` `setJMSCorrelationID(string)`
`CorrelationIDAsBytes`	Correlation ID	`getCorrelationIDAsBytes` `setCorrelationIDAsBytes(byte[])`
`DeliveryMode`	Persistent, nonpersistent (default = persistent)	"getDeliveryMode()" on page 46 "setDeliveryMode(arg0)" on page 62
`Destination`	Destination (default = message destination as configured in Connectivity Map)	"getDestination()" on page 47 "setDestination(arg0)" on page 62
`Expiration`	Number in milliseconds (default = 0 ms = never expire)	"getTimeToLive()" on page 48 "setTimeToLive(arg0)" on page 64

Property	Values	Equivalent JMS Methods
`MessageID`	Message ID	`getJMSMessageID()` `setJMSMessageID(string)`
`Priority`	0 - 9 where 9 is the highest priority (default = 4)	"getPriority()" on page 48 "setPriority(arg0)" on page 63
`Redelivered`	True, false	`getJMSRedelivered()` `setJMSRedelivered(boolean)`
`ReplyTo`	Destination	`getJMSReplyTo()` `setJMSReplyTo(destination)`
`Timestamp`	Number in milliseconds	`getJMSTimestamp()` `setJMSTimestamp(long)`
`Type`	Text, Bytes, Map, Stream	"getJMSMessageType()" on page 67 "setJMSMessageType(arg0)" on page 77

Additional JMS Message Properties

You can also set the message properties for outbound JMS messages. These property nodes are shown in the following figure, as they appear in the user interface. These properties determine only how the message is sent. When you set these properties in a Java based Collaboration Definition, they override the corresponding JMS client message properties, and are used only by the Collaboration that uses that specific Collaboration Definition. For information on setting these properties in the JMS clients, see *Configuring JMS Clients*.

FIGURE 14 JMS OTD Outbound Property Nodes

The following table shows the allowed values for the outbound JMS message properties and the JMS methods used when you set a property in a Collaboration.

Property	Allowed Values	Equivalent JMS Methods
`deliveryMode`	Persistent, nonpersistent	"getDeliveryMode()" on page 46 "setDeliveryMode(arg0)" on page 62
`priority`	0 - 9 where 9 is the highest priority	"getPriority()" on page 48 "setPriority(arg0)" on page 63
`timeToLive`	Number in milliseconds	"getTimeToLive()" on page 48 "setTimeToLive(arg0)" on page 64
`destination`	Destination	"getDestination()" on page 47 "setDestination(arg0)" on page 62
`MessageServerURL`	A valid URL	"getMessageServerURL()" on page 47 "setMessageServerURL(arg0)" on page 63

For more information about these properties, refer to the equivalent JMS method description in the indicated locations.

JMS OTD Methods

This topic describes the Java methods available for use with objects of the type `com.stc.connectors.jms.JMS`, and can be accessed from:

- The JMS OTD
- Input messages for the `send` web service operation
- Output messages for the `receive` web service operation (new web service)

createBytesMessage()

Description

Creates an empty byte message.

Parameters

None.

Return Value

Returns	Type
The byte message object.	`com.stc.connectors.jms.Message`

Exceptions

None.

createBytesMessage(msg)

Description

Creates a byte message with the specified byte array value.

Parameters

Name	Type	Description
msg	`byte[]`	The byte array value for the bytes message

Return Value

Returns	Type
The message object with data *msg*.	`com.stc.connectors.jms.Message`

Exceptions

None.

createMapMessage()

Description

Creates a map message.

Parameters

None.

Return Value

Returns	Type
The `MapMessage` object.	`com.stc.connectors.jms.Message`

Exceptions

None.

createMessage()

Description

Creates an empty message (no payload), which is the most efficient method for event notification.

Parameters

None.

Return Value

Returns	Type
The message object.	`com.stc.connectors.jms.Message`

Exceptions

None.

createMessage(msg)

Description

Creates a message of type `com.stc.connectors.jms.Message`, which wraps around the message variable of type `javax.jms.Message`.

Parameters

Name	Type	Description
msg	`javax.jms.Message`	The message variable.

Return Value

Returns	Type
The message object, with data from JMS Message object.	`com.stc.connectors.jms.Message`

Exceptions

Throws `JMSException`, `IOException`.

createStreamMessage()

Description

Creates an empty stream message. Use the method "setStreamMessage(arg0, arg1)" on page 78 to add the payload.

Parameters

None.

Return Value

Returns	Type
The stream message object.	`com.stc.connectors.jms.Message`

Exceptions

None.

createTextMessage()

Description

Creates an empty text message. Use the method "setTextMessage(arg0)" on page 78 to add the payload.

Parameters

None.

Return Value

Returns	Type
The `TextMessage` object.	`com.stc.connectors.jms.Message`

Exceptions

None.

createTextMessage(msg)

Description

Creates a text message that includes the specified text, producing a ready-to-deliver `TextMessage` object.

Parameters

Name	Type	Description
msg	`String`	The string to populate the `msg` object with.

Return Value

Returns	Type
The message object with data *msg*.	`com.stc.connectors.jms.Message`

Exceptions

None.

getDeliveryMode()

Description

Gets and returns the value of the JMS delivery mode.

Parameters

None.

Return Value

Returns	Type
The value of the JMS `deliveryMode` property.	`java.lang.String`

Exceptions

None.

getDestination()

Description

Gets and returns the name of the queue or topic.

Parameters

None.

Return Value

Returns	Type
The message destination name.	`java.lang.String`

Exceptions

None.

Also Accessible From

- Input messages for the `receiveWait` web service operation.

getMessageServerURL()

Description

Gets and returns the value of the message server URL.

Parameters

None.

Return Value

Returns	Type
The value of the message server URL.	`java.lang.String`

Exceptions

None.

getPriority()

Description

Gets and returns the value of the JMS priority.

Parameters

None.

Return Value

Returns	Type
The value of the JMS `priority` property.	`int`

Exceptions

None.

getTimeToLive()

Description

Gets and returns the specified time period, in milliseconds, following the dispatch time that a message should be retained by the message system.

Parameters

None.

Return Value

Returns	Type
The default time period of the JMS `timeToLive` property, in milliseconds.	`long`

Exceptions

None.

receive(timeout)

Description

Receives the next message of type `com.stc.connectors.jms.Message` that arrives from the destination configured in the Connectivity Map Editor during the specified timeout interval.

Parameters

Name	Type	Description
timeout	`long`	The number of milliseconds before the `receive` method times out.

Return Value

Returns	Type
The next message produced, or null if the timeout expires.	`com.stc.connectors.jms.Message`

Exceptions

Throws `JMSException`.

receive(timeout, destination)

Description

Receives the next message of type `com.stc.connectors.jms.Message` that arrives from the specified message destination during the specified timeout interval.

Parameters

Name	Type	Description
timeout	`long`	The number of milliseconds before the `receive` method times out.
destination	`java.lang.String`	The name of the topic or queue from which this method receives messages.

Return Value

Returns	Type
The next message produced, or null if the timeout expires.	`com.stc.connectors.jms.Message`

Exceptions

Throws `JMSException`.

receiveNoWait()

Description

Receives the next message of type `com.stc.connectors.jms.Message`, if one is immediately available.

Parameters

None.

Return Value

Returns	Type
The next message produced, or null if no message is available.	`com.stc.connectors.jms.Message`

Exceptions

Throws `JMSException`.

receiveNoWait(destination)

Description

Receives the next message of type `com.stc.connectors.jms.Message` from the specified message destination if a message is immediately available.

Parameters

Name	Type	Description
destination	`java.lang.String`	The name of the topic or queue from which this method receives messages.

Return Value

Returns	Type
The next message produced, or null if no message is available.	`com.stc.connectors.jms.Message`

Exceptions

Throws `JMSException`.

requestReply(message)

Description

Sends a message of type `com.stc.connectors.jms.Message` to the destination configured in the Connectivity Map Editor, and waits for the reply message by using `Topic/QueueRequestor`.

Parameters

Name	Type	Description
message	`com.stc.connectors.jms.Message`	The message object to send

Return Value

Returns	Type
The reply message.	`com.stc.connectors.jms.Message`

Exceptions

Throws `JMSException` when a message is null or when the JMS provider fails to send and receive the message due to an internal error.

requestReply(timeout, message)

Description

Sends a message of type `com.stc.connectors.jms.Message` to the destination configured in the Connectivity Map Editor, and receives the reply message by using `Topic/QueueRequestor` during the specified timeout interval.

Parameters

Name	Type	Description
timeout	`java.lang.long`	The timeout in milliseconds
message	`com.stc.connectors.jms.Message`	The message object to send

Return Value

Returns	Type
The reply message, if available during the timeout interval.	`com.stc.connectors.jms.Message`

Exceptions

Throws `JMSException` if a message is null, the JMS provider fails to send and receive the message due to an internal error, or timeout expires.

requestReplyTo(message, destName)

Description

Sends a message of type `com.stc.connectors.jms.Message` to the explicitly named destination, and waits for the reply message by using `Topic/QueueRequestor`.

Parameters

Name	Type	Description
message	`com.stc.connectors.jms.Message`	The message object to send
destName	`java.lang.String`	The destination name

Return Value

Returns	Type
The reply message.	`com.stc.connectors.jms.Message`

Exceptions

Throws `JMSException` when a message is null, *destName* is null, or the JMS provider fails to send and receive the message due to an internal error.

requestReplyTo(timeout, message, destName)

Description

Sends a message of type `com.stc.connectors.jms.Message` to the explicitly named destination, and receives the reply message by using `Topic/QueueRequestor`.

Parameters

Name	Type	Description
timeout	`java.lang.long`	The timeout in milliseconds
message	`com.stc.connectors.jms.Message`	The message object to send
destName	`java.lang.String`	The destination name

Return Value

Returns	Type
The reply message, if available during the timeout interval.	`com.stc.connectors.jms.Message`

Exceptions

Throws `JMSException` when a message is null, *destName* is null, when the JMS provider fails to send and receive the message due to an internal error, or timeout expires.

send(message)

Description

Sends a message of type `com.stc.connectors.jms.Message` to the destination configured in the Connectivity Map Editor, using the JMS session's default settings for message priority, time to live, and delivery mode.

Parameters

Name	Type	Description
message	`com.stc.connectors.jms.Message`	The message variable type to be sent.

Return Value

None.

Exceptions

Throws `JMSException` if a message is null or if the JMS provider fails to send the message due to an internal error.

send(message, deliveryMode, priority, timeToLive)

Description

Sends a message of type `com.stc.connectors.jms.Message` to the destination configured in the Connectivity Map Editor, using the values specified for message priority, time to live, and delivery mode.

Parameters

Name	Type	Description
message	`com.stc.connectors.jms.Message`	The message variable to be created.
deliveryMode	`int`	The message delivery mode. 1 indicates nonpersistent messages, 2 indicates persistent messages.
priority	`int`	The message priority (0 through 9, with 9 being the highest priority).
timeToLive	`long`	The length of time in milliseconds before the message expires.

Return Value

None.

Exceptions

Throws `JMSException` if a message is null or if the JMS provider fails to send the message due to an internal error.

sendBytes(payload)

Description

Sends a message of type `byte[]` to the destination configured in the Connectivity Map Editor, using the JMS session's default settings for message priority, time to live, and delivery mode.

Parameters

Name	Type	Description
payload	`byte[]`	The message byte array value.

Return Value

None.

Exceptions

Throws `JMSException` if a message is null or if the JMS provider fails to send the message due to an internal error.

sendBytes(payload, deliveryMode, priority, timeToLive)

Description

Sends a message of type `byte[]` to the destination configured in the Connectivity Map Editor, using the specified parameters.

Parameters

Name	Type	Description
payload	`byte[]`	The byte array value.
deliveryMode	`int`	The message delivery mode. 1 indicates nonpersistent messages, 2 indicates persistent messages.
priority	`int`	The message priority (0 through 9, with 9 being the highest priority).
timeToLive	`long`	The length of time in milliseconds before the message expires.

Return Value

None.

Exceptions

Throws `JMSException` if a message is null or if the JMS provider fails to send the message due to an internal error.

sendBytesTo(payload, destination)

Description

Sends a message of type `byte[]` to the specified destination, using the JMS session's default settings for message priority, time to live, and delivery mode.

Parameters

Name	Type	Description
payload	`byte[]`	The byte array value.
destination	`java.lang.String`	The name of the topic or queue from which this method receives messages.

Return Value

None.

Exceptions

Throws `JMSException` if a message is null or if the JMS provider fails to send the message due to an internal error.

sendBytesTo(payload, destination, deliveryMode, priority, timeToLive)

Description

Sends a message of type `byte[]` to the destination configured in the Connectivity Map Editor, using the specified parameters.

Parameters

Name	Type	Description
payload	`byte[]`	The byte array value.
deliveryMode	`int`	The message delivery mode. 1 indicates non-persistent messages, 2 indicates persistent messages.
priority	`int`	The message priority (0 through 9, with 9 being the highest priority).
timetolive	`long`	The amount in milliseconds before the message expires.

Return Value

None.

Exceptions

Throws `JMSException` if a message is null or if the JMS provider fails to send the message due to an internal error.

sendText(payload)

Description

Sends a message of type `java.lang.String` to the destination configured in the Connectivity Map Editor, using the JMS session's default settings for message priority, time to live, and delivery mode.

Parameters

Name	Type	Description
payload	`java.lang.String`	The text in the message.

Return Value

None.

Exceptions

Throws `JMSException` if a message is null or if the JMS provider fails to send the message due to an internal error.

sendText(payload, deliveryMode, priority, timeToLive)

Description

Sends a message of type `java.lang.String` to the destination configured in the Connectivity Map Editor, using the specified parameters.

Parameters

Name	Type	Description
payload	`java.lang.String`	The text in the message.
deliveryMode	`int`	The message delivery mode. 1 indicates non-persistent messages, 2 indicates persistent messages.
priority	`int`	The message priority (0 through 9, with 9 being the highest priority).
timeToLive	`long`	The length of time in milliseconds before the message expires.

Return Value

None.

Exceptions

Throws `JMSException` if a message is null or if the JMS provider fails to send the message due to an internal error.

sendTextTo(payload, destination)

Description

Sends a message of type `java.lang.String` to the specified destination, using the JMS session's default settings for message priority, time to live, and delivery mode.

Parameters

Name	Type	Description
payload	`java.lang.String`	The text in the message.
destination	`java.lang.String`	The name of the topic or queue to which this method should send the message.

Return Value

None.

Exceptions

Throws `JMSException` if a message is null or if the JMS provider fails to send the message due to an internal error.

sendTextTo(payload, destination, deliveryMode, priority, timeToLive)

Description

Sends a message of type `java.lang.String` to the specified destination, using the specified parameters.

Parameters

Name	Type	Description
payload	`java.lang.String`	The text in the message.
destination	`java.lang.String`	The name of the topic or queue to which this method should send the message.
deliveryMode	`int`	The message delivery mode. 1 indicates non-persistent messages, 2 indicates persistent messages.
priority	`int`	The message priority (0 through 9, with 9 being the highest priority).
timeToLive	`long`	The length of time in milliseconds before the message expires.

Return Value

None.

Exceptions

Throws `JMSException` if a message is null or if the JMS provider fails to send the message due to an internal error.

sendTo(message, destination)

Description

Sends a message of type `com.stc.connectors.jms.Message` to the specified destination, using the JMS session's default settings for message priority, time to live, and delivery mode.

Parameters

Name	Type	Description
msg	`com.stc.connectors.jms.Message`	The message to be sent.
destination	`java.lang.String`	The name of the topic or queue to which this method should send the message.

Return Value

None.

Exceptions

Throws `JMSException` if a message is null or if the JMS provider fails to send the message due to an internal error.

sendTo(message, destination, deliveryMode, priority, timeToLive)

Description

Sends a message of type `com.stc.connectors.jms.Message` to the specified destination, using the specified parameters.

Parameters

Name	Type	Description
message	`com.stc.connectors.jms.Message`	The message variable type.
destination	`java.lang.String`	The name of the topic or queue from which this method receives messages.

Name	Type	Description
deliveryMode	`int`	The message delivery mode. 1 indicates non-persistent messages, 2 indicates persistent messages.
priority	`int`	The message priority (0 through 9, with 9 being the highest priority).
timeToLive	`long`	The length of time in milliseconds before the message expires.

Return Value

None.

Exceptions

Throws `JMSException` if a message is null or if the JMS provider fails to send the message due to an internal error.

setDeliveryMode(arg0)

Description

Sets the value of the JMS delivery mode.

Parameters

Name	Type	Description
arg0	`java.lang.String`	The value of the JMS `deliveryMode` property.

Return Value

None.

Exceptions

None.

setDestination(arg0)

Description

Specifies the queue or topic.

Parameters

Name	Type	Description
arg0	`java.lang.String`	The message destination name.

Return Value

None.

Exceptions

None.

Also Accessible From

- Input messages for the `receiveWait` web service operation.

setMessageServerURL(arg0)

Description

Sets the value of the message server URL.

Parameters

Name	Type	Description
arg0	`java.lang.String`	The value of the message server URL.

Return Value

None.

Exceptions

None.

setPriority(arg0)

Description

Sets the value of the JMS priority.

Parameters

Name	Type	Description
arg0	`int`	The value of the JMS `priority` property.

Return Value

None.

Exceptions

None.

setTimeToLive(arg0)

Description

Sets the default time period, in milliseconds, following the dispatch time that a message should be retained by the message system.

Parameters

Name	Type	Description
arg0	`long`	The default time period for the JMS `timeToLive` property, in milliseconds

Return Value

None.

Exceptions

None.

JMS Message Methods

This topic describes Java methods that are available for messages of the type `com.stc.connectors.jms.Message`, and can be accessed from:

- Input messages for the `receive` web service operation.
- Output messages for the `receiveWait` web service operation.

The following Java methods are available for messages of the type `com.stc.connectors.jms.receiveWaitMessage`, and can be accessed from the `receiveWait` web service operation.

countMapMessage()

Description

Counts the number of keys in the map message.

Parameters

None.

Return Value

Returns	Type
The number of keys in the map message.	`int`

Exceptions

None.

countStreamMessage()

Description

Counts the number of items in the map message.

Parameters

None.

Return Value

Returns	Type
The number of items in the stream message.	`int`

Exceptions

None.

countUserProperty()

Description

Counts the number of user properties.

Parameters

None.

Return Value

Returns	Type
The number of user properties.	`int`

Exceptions

None.

getBytesMessage()

Description

Gets the data bytes in the message.

Parameters

None.

Return Value

Returns	Type
A byte array representing the data in the message.	`byte []`

Exceptions

None.

getJMSMessageType()

Description

Gets the JMS message type.

Parameters

None.

Return Value

Returns	Type
The JMS message type, for example, *Bytes*, *Text*, *Stream*, *Map*.	`java.lang.String`

Exceptions

None.

getMapMessage()

Description

Gets the name-value pair array containing the map message.

Parameters

None.

Return Value

Returns	Type
The `NameValuePair` array containing the map message.	`com.stc.connectors.jms.NameValuePair`

Exceptions

None.

getMapMessage(arg0)

Description

Gets the name-value pair containing the map message from the location in the map list specified by *arg0*.

Parameters

Name	Type	Description
arg0	`int`	The location of the name-value pair in the map message list.

Return Value

Returns	Type
The `NameValuePair` object at the specified location.	`com.stc.connectors.jms.NameValuePair`

Exceptions

None.

getMessageProperties()

Description

Gets the message property object to query for the various JMS message properties.

Parameters

None.

Return Value

Returns	Type
The `MessageProperty` object to query for the various JMS message properties.	`com.stc.connectors.jms.MessageProperty`

Exceptions

None.

getStreamMessage()

Description

Gets the array of stream message objects.

Parameters

None.

Return Value

Returns	Type
An array of stream message objects.	`java.lang.Object`

Exceptions

None.

getStreamMessage(arg0)

Description

Gets the stream message object from the location specified by *arg0*.

Parameters

Name	Type	Description
arg0	`int`	The location of the stream message in the list.

Return Value

Returns	Type
The `StreamMessage` object at the specified location.	`java.lang.Object`

Exceptions

None.

getTextMessage()

Description

Gets the data string in the message.

Parameters

None.

Return Value

Returns	Type
The data string in the message.	`java.lang.String`

Exceptions

None.

getUserProperty()

Description

Gets the name-value pair array of the user properties in the message.

Parameters

None.

Return Value

Returns	Type
The `NameValuePair` array of the user properties in the message.	`com.stc.connectors.jms.NameValuePair`

Exceptions

None.

getUserProperty(arg0)

Description

Gets the name-value pair representing user properties from the location specified by *arg0*.

Parameters

Name	Type	Description
arg0	`int`	Location of the user property in the list.

Return Value

Returns	Type
The `NameValuePair` object from the specified location.	`com.stc.connectors.jms.NameValuePair`

Exceptions

None.

retrieveBytesFromMessage()

Description

Returns the byte array corresponding to the payload of the message object.

Parameters

None.

Return Value

Returns	Type
The byte array corresponding to the payload of the message object.	`byte[]`

Exceptions

Throws `JMSException` if conversion is not possible, for example, if the method cannot convert from a map message to a byte array.

retrieveBytesFromMessage(arg0)

Description

Returns the byte array corresponding to the message payload, with the encoding as specified by *arg0*.

Parameters

Name	Type	Description
arg0	`java.lang.String`	The encoding to use when converting to a byte array.

Return Value

Returns	Type
The byte array corresponding to the message payload, with the specified encoding.	`byte[]`

Exceptions

- Throws `JMSException` if conversion is not possible, for example, if the method cannot convert from a map message to a byte array.
- Throws `UnsupportedEnclodingException` if the name `charset` is not supported.

retrieveMapMessage(arg0)

Description

Returns the value of the message specified by *arg0*, or null if the message does not exist.

Parameters

Name	Type	Description
arg0	`java.lang.String`	The name of the map message.

Return Value

Returns	Type
The value of the specified message, or null if the message does not exist.	`java.lang.Object`

Exceptions

None.

retrieveMapMessageList()

Description

Returns a map message list object, which contains map message objects as an array of name-value pairs.

Parameters

None.

Return Value

Returns	Type
A `MapMessageList` object, which contains map message objects as name-value pairs.	`com.stc.connectors.jms.MapMessageList`

Exceptions

None.

retrieveStringFromMessage()

Description

Returns a string representation of the message payload.

Parameters

None.

Return Value

Returns	Type
A string corresponding to the message payload.	`java.lang.String`

Exceptions

Throws `JMSException` if conversion is not possible, for example, if the method cannot convert from a map message to a String.

retrieveStringFromMessage(arg0)

Description

Returns a string representation of the message payload, with the encoding as specified by *arg0*.

Parameters

Name	Type	Description
arg0	`java.lang.String`	The encoding to use when converting to a string object.

Return Value

Returns	Type
The string corresponding to the message payload, with the specified encoding.	`java.lang.String`

Exceptions

Throws `JMSException` if conversion is not possible.

retrieveUserProperty(arg0)

Description

Returns the value of the user-defined string property specified by *arg0*, or null if the property does not exist.

Parameters

Name	Type	Description
arg0	`java.lang.String`	The name of the user property.

Return Value

Returns	Type
The value of the specified user property, or null if the property does not exist.	`java.lang.String`

Exceptions

None.

retrieveUserPropertyList()

Description

Returns the user property list.

Parameters

None.

Return Value

Returns	Type
A `UserPropertyList` object, which contains user properties as name-value pairs.	`com.stc.connectors.jms.UserPropertyList`

Exceptions

None.

setBytesMessage(arg0)

Description

Sets the bytes message to the value specified by *arg0*.

Parameters

Name	Type	Description
arg0	`byte[]`	The byte array containing the message.

Return Value

None.

Exceptions

None.

setJMSMessageType(arg0)

Description

Sets the text message type to the value specified by *arg0*.

Parameters

Name	Type	Description
arg0	`java.lang.String`	The value to which to set the JMS message type, for example, *Bytes*, *Text*, *Stream*, *Map*.

Return Value

None.

Exceptions

None.

setStreamMessage(arg0, arg1)

Description

Sets the stream message specified by *arg0* to the value specified by *arg1*.

Parameters

Name	Type	Description
arg0	`int`	The index of the stream message to be set.
arg1	`java.lang.Object`	The value to which to set the stream message.

Return Value

None.

Exceptions

None.

setTextMessage(arg0)

Description

Sets the text message to the value specified by *arg0*.

Parameters

Name	Type	Description
arg0	`java.lang.String`	The value to which to set the text message.

Return Value

None.

Exceptions

None.

storeMapMessage(arg0, arg1)

Description

Writes the name and value of a map message to the map message object, where *arg0* specifies the name and *arg1* specifies the value of the map message.

Parameters

Name	Type	Description
arg0	`java.lang.String`	The name of the map message.
arg1	`java.lang.Object`	The value of the map message.

Return Value

None.

Exceptions

None.

storeUserProperty(arg0, arg1)

Description

Writes the name and value of a user property to the user property object, where *arg0* specifies the name and *arg1* specifies the value of the user property.

Parameters

Name	Type	Description
arg0	`java.lang.String`	The name of the user property.
arg1	`java.lang.String`	The value of the user property.

Return Value

None.

Exceptions

None.

getTimeToWait()

Description

Gets and returns the timeout period during which the operation blocks program execution, while waiting for a message to arrive.

Parameters

None.

Return Value

Returns	Type
The time to wait for a message, in milliseconds.	`long`

Exceptions

None.

setTimeToWait(arg0)

Description

Sets the timeout period during which the operation blocks program execution, while waiting for a message to arrive.

Parameters

Name	Type	Description
arg0	`long`	The time, in milliseconds, for the operation to block program execution.

Return Value

None.

Exceptions

None.

JMS Client Configuration

This topic describes the configuration properties for JMS Client connectors.

Categories

The following tables list the JMS client configuration properties as displayed in the NetBeans IDE configuration dialogs. These tables provide a cross-reference to the property descriptions in the following section, which are categorized by *consumer* and *producer*, and presented alphabetically.

TABLE 7 Root Properties

Property	Applies to	Description
`Durable Subscriber Name`	Topic subscribers	"Durable Subscriber Name" on page 84

TABLE 8 Basic Properties

Property	Applies to	Description
`Concurrency`	Topic subscribers Queue receivers	"Concurrency" on page 83
`Delivery mode`	Topic publishers Queue senders	"Delivery Mode" on page 87

TABLE 8 Basic Properties *(Continued)*

Property	Applies to	Description
`Idle timeout`	Topic publishers Queue senders	"Idle Timeout" on page 88
`Maximum pool size`	Topic publishers Queue senders	"Maximum Pool Size" on page 88
`Maximum wait time`	Topic publishers Queue senders	"Maximum Wait Time" on page 88
`Message selector`	Topic subscribers Queue receivers	"Message Selector" on page 85
`Priority`	Topic publishers Queue senders	"Priority" on page 89
`Steady pool size`	Topic publishers Queue senders	"Steady Pool Size" on page 89
`Transaction mode`	Topic publishers Queue senders	"Transaction Mode" on page 89

TABLE 9 Redelivery Handling Properties

Property	Applies to	Description
`Delay`	Topic subscribers Queue receivers	"Delay" on page 84
`Move/Delete After N Times`	Topic subscribers Queue receivers	"Move/Delete After N Times" on page 86
`Action`	Topic subscribers Queue receivers	"Action" on page 83
`Move to Queue/Topic`	Topic subscribers Queue receivers	"Move to Queue/Topic" on page 86
`Move to Destination Name`	Topic subscribers Queue receivers	"Move to Destination Name" on page 86

TABLE 10 Advanced Properties

Property	Applies to	Description
`Durability`	Topic subscribers	"Durability" on page 85
`Server session batch size`	Topic subscribers Queue receivers	"Server Session Batch Size" on page 87
`Server session pool size`	Topic subscribers Queue receivers	"Server Session Pool Size" on page 87

The sections below describe the JMS client connector properties in detail, and supplement the information given in *Configuring JMS Clients*.

Consumers

Action

The `Action` property specifies the action to take following a specified number of unsuccessful attempts to deliver a message. This property specifies whether to `move` (redirect) or `delete` the message after the number of retries specified in "Delay" on page 84. See "Message Redelivery and Redirection" on page 26 for additional information.

Allowed Values

The allowed values are `move`, `delete`, and `no final action`.

Default Value

The default is `no final action`, which specifies continued retries until the message is received.

Concurrency

The `Concurrency` property specifies whether the message consumers use *connection consumer* or *serialized* processing. To use concurrent processing for a connection, select the `Connection consumer` setting. To use serial execution, select the `Serial mode` setting. This property applies to topic subscribers and queue receivers, and is specified in the *Basic* properties dialog.

You can configure JMS clients to use connection consumers to improve message throughput through concurrent processing. Connection consumers consume messages that are specified by a destination and an optional message selector. See "Message Selector" on page 85.

To start processing, a connection consumer gets a server session from its pool and loads the session with a message. Server sessions associate a JMS session with a thread. The server session pool is a set of server sessions provided to a connection consumer to process its messages.

The use of connection consumers increases message processing performance by enabling concurrent processing using multiple threads. You can specify the number of message driven beans (MDBs) or server session pool to assign to a JMS Collaboration to process messages concurrently. When you use connection consumer with fully concurrent or protected concurrent FIFO processing, this setting allows the integration server to assign multiple threads to execute the Collaboration on a particular message destination.

For queues, you can also use connection consumers for concurrent processing on multiple CPUs and application servers on a single system. This configuration does affect FIFO processing. For information, refer to "Message Processing Order" on page 20.

You specify the maximum number of threads per server session pool as described in "Server Session Pool Size" on page 87. By default, the maximum number of threads is `5`.

The maximum number of messages that a connection consumer can load into a server session at one time is set at `1` and cannot be changed (see "Server Session Batch Size" on page 87).

Default Value

The default JMS client concurrency mode is `Serial mode`.

Delay

The `Delay` property specifies the delay(s) to apply following a specified number of unsuccessful attempts to deliver a message. The format is *retries:delay*, where the number of retries is counted from the original rollback and the delay time is in milliseconds. Progressive delays can be specified by concatenating *retry:delay* pairs separated by a comma and a space:

retry`:`*delay*`, `*retry*`:`*delay*`, ..., `*retry*`:`*delay*

This property applies to topic subscribers and queue receivers, and is specified in the `Redelivery Handling` properties dialog. See "Message Redelivery and Redirection" on page 26 for additional information.

Allowed Values

The maximum allowed delay is five seconds (`5000` ms).

Default Value

By default, no value is specified.

Durable Subscriber Name

The `Durable Subscriber Name` property, which is specified in the root *JMS Client* properties dialog, both provides a name for the JMS client and identifies it as being a durable subscriber.

A durable subscription is one that is not dependent upon a client's connection with a message server. Therefore, it is tolerant of disconnections, whether they are intentional or not. When a durable subscriber is disconnected from the message server, the server stores messages until the subscriber reconnects. It then delivers all accumulated messages that have not expired. This process is also known as *store-and-forward* messaging.

Note – Once set, the Durable Subscriber Name does not get autogenerated, and can only be changed manually. Copies of the Connectivity Map also retain this name.

Durability

The `Durability` property specifies whether or not the subscriber to this JMS connection is durable. When a subscriber is nondurable, the client sees messages on a topic only when the subscriber is active. If messages are published when the subscriber is inactive, the messages are lost. This property applies to topic subscribers only, and is specified in the *Basic* properties dialog.

When a subscriber is durable, messages are not lost even when the subscriber is inactive because the message server retains the messages until they are retrieved by the subscriber or until the messages expire. A durable subscriber registers with the message server as a durable subscriber with the name *source_destination*, for example, `topicA_CollaborationA`. When a subscriber becomes inactive, the message server retains the unexpired messages for a subsequent subscriber object with the same identity to resume the subscription. Note that there is a trade-off in performance.

Allowed Values

`Durable` or `Nondurable`.

Default Value

By default, JMS client connections are `Durable`.

Message Selector

The `Message selector` property specifies message selectors for the JMS client. This property applies to topic subscribers and queue receivers, and is specified in the *Basic* properties dialog.

To specify a message selector, enter a message selector String according to the JMS specification syntax. For example:

```
JMSType ='car' AND color ='blue'
```

Identifier start characters must be characters for which the `Character.isJavaIdentifierStart` method returns `true` according to the JMS specification. If the identifier is invalid, the Java CAPS Monitor might show invalid message property names.

Move/Delete After N Times

The `Move/Delete After N Times` property specifies the number of retries to allow before redirecting or deleting the message, as specified in "Action" on page 83. The action is taken on the retry number entered for the property value.

This property applies to topic subscribers and queue receivers, and is specified in the *Redelivery Handling* properties dialog. See "Message Redelivery and Redirection" on page 26 for additional information.

Allowed Values

Any number.

Default Value

By default, no value is specified.

Move to Queue/Topic

The `Move to Queue/Topic` property specifies whether to redirect the message to a queue or a topic, following the number of retries specified in "Move/Delete After N Times" on page 86.

This property applies to topic subscribers and queue receivers, and is specified in the *Redelivery Handling* properties dialog. See "Message Redelivery and Redirection" on page 26 for additional information.

Allowed Values

The allowed values are `queue`, `topic`, and `auto`, which specifies the same kind of message destination as the message producer.

Default Value

By default, the destination is specified as `auto`.

Move to Destination Name

The `Move to Destination Name` property specifies a queue or topic name to which the message is to be redirected. The special character `$` specifies the original destination name.

This property applies to topic subscribers and queue receivers, and is specified in the *Redelivery Handling* properties dialog. See "Message Redelivery and Redirection" on page 26 for additional information.

Allowed Values

Any string.

Default Value

By default, no value is specified.

Server Session Batch Size

The `Server session batch size` property specifies the maximum number of messages that a connection consumer can load into a server session at one time. This property applies to topic subscribers and queue receivers, and is specified in the *Advanced* properties dialog. By default, this property is set to **1** and cannot be changed.

Server Session Pool Size

The `Server session pool size` property specifies the maximum number of threads per ServerSessionPool to be used for concurrent processing. This property applies to topic subscribers and queue receivers, and is specified in the *Advanced* properties dialog.

This property is used in conjunction with the connection consumer setting of the `Concurrency` property, as described in "Concurrency" on page 83). You can specify the number of message driven beans (MDBs) or server session pool to assign to a JMS Collaboration to process messages concurrently. When you use connection consumer with fully concurrent or protected concurrent FIFO processing, this connection consumer configuration allows the integration server to assign multiple threads to execute the Collaboration on a particular message destination.

For an overview about message processing, refer to "Message Processing Order" on page 20.

Allowed Values

An integer of **1** or larger, depending on the capability of the system, indicating the number of threads.

Default Value

By default, the maximum number of threads per server session pool is `5`.

Producers

Delivery Mode

The `Delivery mode` property specifies whether the messages for this JMS connection are persistent or non-persistent. This property applies to topic publishers and queue senders, and is specified in the *Basic* properties dialog.

Non-persistent delivery mode is the most efficient delivery mode, because it does not require messages to be saved to permanent storage. Per JMS specification, the message destination delivers non-persistent messages with an at-most-once guarantee. The message is only delivered once, even if it is lost. This mode involves a trade-off between performance and reliability. Non-persistence offers better performance but if a message server fails, messages may be lost due to a power outage.

When messages are persistent, the message server places the message in permanent storage to ensure the message is not lost in transit if the message server fails. Persistent messages are delivered once, and only once.

For the JMS IQ Manager, persistent messages are stored in the message server database files.

Default Value

The default delivery mode is **Persistent**.

Idle Timeout

The `Idle timeout` property specifies the amount of time, in seconds, to wait before returning a connection to the pool. This property applies to topic publishers and queue senders, and is specified in the *Basic* properties dialog.

Default Value

The default timeout is `30` seconds.

Maximum Pool Size

The `Maximum pool size` property specifies the maximum number of connections to be made to the message server. This property applies to topic publishers and queue senders, and is specified in the *Basic* properties dialog.

Default Value

The default size is `32`.

Maximum Wait Time

The `Maximum wait time` property specifies the maximum amount of time, in milliseconds, to wait for acquiring a connection before throwing an exception. This property applies to topic publishers and queue senders, and is specified in the *Basic* properties dialog.

Default Value

The default time is `30000` milliseconds.

Priority

The `Priority` property specifies the message priority level for the JMS client. The message priority level that you specify causes all messages produced by this client to have that same priority level. For example, if you set the priority level to 2, all messages sent by that client have message priority level 2. This property applies to topic publishers and queue senders, and is specified in the *Basic* properties dialog.

You can also specify message priorities in Collaborations with the JMS OTD with the *setPriority* method. Collaboration message priorities override JMS client message priorities. For more information, refer to *Configuring JMS Clients*.

Allowed Values

An integer between `0` and `9`, where 0 through 4 is normal priority and 5 through 9 is expedited priority.

Default Value

The default delivery mode is `4`.

Steady Pool Size

The `Steady pool size` property specifies the minimum, and initial, number of connections maintained in the pool. This property applies to topic publishers and queue senders, and is specified in the *Basic* properties dialog.

Default Value

The default is `4` connections.

Transaction Mode

The `Transaction mode` property specifies the transaction mode used for message producers. For consumers, this mode is always `XA`. This property applies to topic publishers and queue senders, and is specified in the *Basic* properties dialog.

The `Transaction mode` property specifies whether messages for this JMS client use one of the following transaction modes:

- **Transacted Mode**

  When you set the transaction mode to `Transacted`, the Java Collaboration Definition uses a new, separate transacted JMS session to send and receive messages. The message is committed automatically, and cannot be rolled back (even by raising an exception).

- **XA Mode**

When the transaction mode is set to XA, the JMS session becomes part of the existing transaction, and is processed according to the XA two-phase commit protocol. In the first phase, the resource manager sends a query to commit to the receivers and waits for the receivers to respond with a confirmation. In the second phase, the resource manager receives confirmation from all receivers, and commits the message to all receivers. This setting prevents message loss and duplicate messages, even when a system unexpectedly shuts down.

Allowed Values

`Transacted` or XA.

Default Value

The default transaction mode is XA.

Note – Documentation on distributed transaction processing using XA is available at no charge from The Open Group at `http://www.opengroup.org` (search on "XA").

JMS IQ Manager Runtime Configuration

This topic provides describes how to configure the runtime properties for JMS IQ Managers.

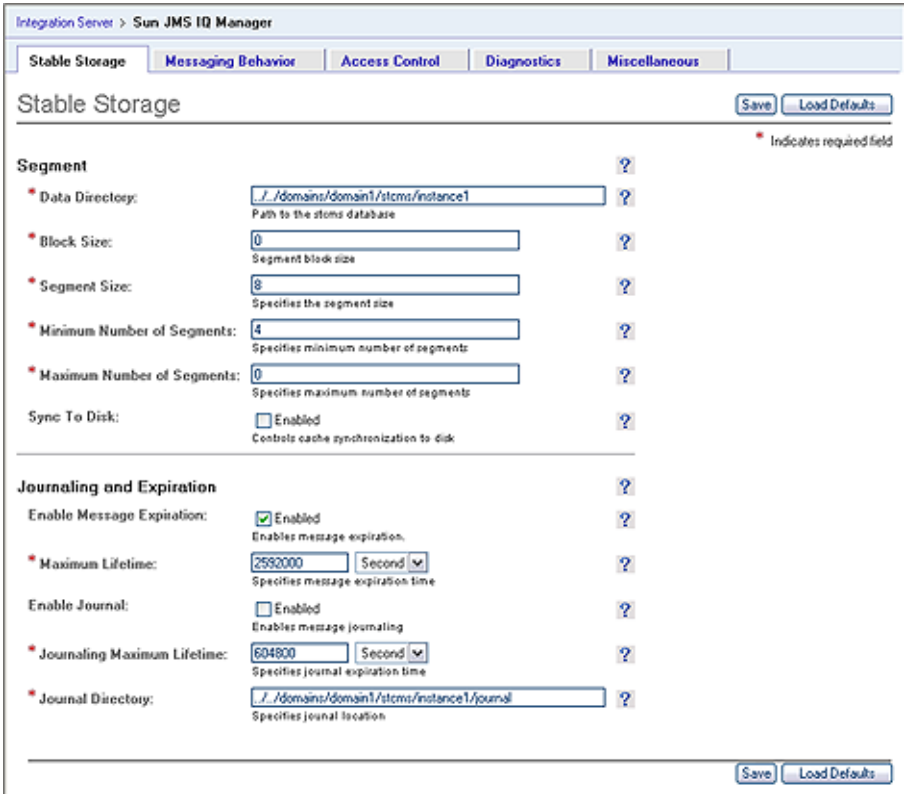
Note – Procedures for configuring properties related to the generation of application files, a design-time issue, are described in *Deploying Java CAPS Projects* .

Accessing the Configuration Properties

The runtime configuration properties for JMS IQ Managers are accessed from the Enterprise Manager's *Application Server Administration* application. Alternate procedures for opening this application are described in *Using Enterprise Manager Management Application in Java CAPS*. Once the application is open, clicking `Sun JMS IQ Manager` in the explorer panel displays a set of configuration pages for the JMS IQ Manager, as shown in the following figures.

FIGURE 15 Application Server Administration Explorer Panel

FIGURE 16 Configuring Runtime JMS IQ Managers



Button/Icon	Function
Save	Clicking this button saves your changes to the page. You must save your changes before proceeding to another page, or the changes will be lost.

Button/Icon	Function
Load Defaults	Clicking this button replaces all properties on the page with their default values.
?	Moving the cursor over this icon displays a description of the specific property or category.

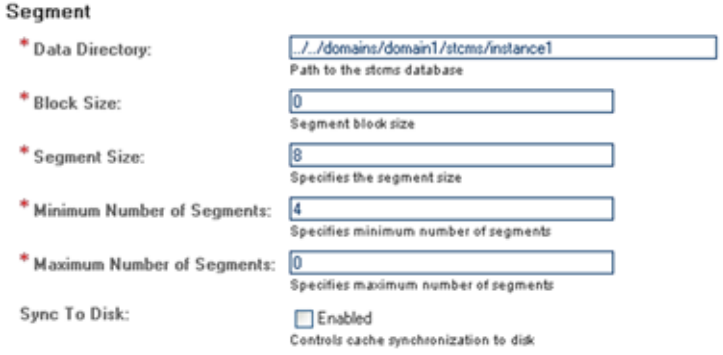
Configuration Properties Interface

Stable Storage Page

Note – Modifying some properties, such as Block Size and Segment Size, requires changes to the database files. In such cases, you must manually delete the database files and restart the domain server before the changes take place. To prevent message loss, make sure that no unread messages remain before deleting the database files.

Segment Properties

FIGURE 17 Segment Properties Panel



As described in "JMS IQ Manager Database" on page 17, the JMS IQ Manager uses the JMS IQ Manager database to store persistent messages. It also stores messages that are larger than can be kept in the JMS IQ Manager memory, which is determined by the cache size setting (by default 0.5 MBytes for Windows and 1 MByte for UNIX). You can specify several properties of this database file, as described in the following sections:

You can also specify whether or not the JMS IQ Manager controls the cache synchronization to disk by using the "Sync to Disk" on page 96 option.

Data Directory

The `Data Directory` property specifies where the JMS IQ Manager database files are located for JMS IQ Managers. You can specify the location as an absolute path, or as a path relative to the `..\logicalhost\is\domains` directory. Using an absolute path for the data directory enables you the store the JMS IQ Manager files on a different system.

If journaling is enabled, the data directory contains a `journal` directory, unless another location has been specified for the `Journal Directory` property. The journal directory holds the journaling database files. For information, refer to *Journal Directory*. Journaling is *disabled* by default.

Default Location

The default setting is: `..\logicalhost\is\domains\domain_1\stcms\instance_1`

Block Size

Data is read from and written to disk in units known as *blocks*. The `Block Size` property specifies the number of bytes per block.

Allowed Values

You can specify `0`, `512`, or `1024` bytes per block. If you specify `0`, the server automatically determines the value by querying the operating system.

Default Value

The default is `0` bytes per block.

Note – Modifying the Block Size requires changes to the database files. You must manually delete the database files and restart the domain server before the changes take place. To prevent message loss, make sure that no unread messages remain before deleting the database files.

Segment Size

The JMS IQ Manager database consists of multiple memory-mapped database files known as *segments*. The `Segment Size` property specifies the total number of pages in each segment file. A page is 512 bytes on Windows and 1024 bytes on UNIX. The default segment size is 16,384 pages, which is 8 MBytes for Windows and 16 MBytes for UNIX. By default, these segments are named `stcms*.dbs` and reside in the message server folder on the Logical Host.

You should set the segment size to a value larger than the sum of the following items:

- The anticipated number of subscribers
- The anticipated maximum transaction size in bytes divided by the page size in bytes
- An additional ten pages to allow for overhead

Note – The transaction size is the sum of the sizes of all messages in one transaction. If transactions span no more than one message, the maximum transaction size is equal to the size of the largest message.

For example, consider a UNIX system, where you expect no more than 100 subscribers and that messages will not exceed 100 KBytes, and that only one message will be sent/received per transaction. Since the page size on UNIX is 1 KByte, you would set the segment size to at least [100 + (100KBytes/1KByte) + 10] = 210 pages.

With this setting, only one 100,000 byte message may exist in each segment. The ideal segment size depends on the circumstances. If the slowest subscriber lags behind the fastest publisher by a certain number of messages, you can set the segment size so that this number of messages will fit a single segment.

The JMS IQ Manager cleans up the database by recycling segments for which all messages have either expired or have been retrieved by their subscribers.

A lower segment size setting results in more efficient use of the disk because smaller segments turn over more rapidly and thus provide more effective use of server memory. However, a lower segment size means that the server might need to allocate more new segments, which requires more time than freeing a cleaned-up segment. In addition, if a transaction is larger than the specified segment size, the server rolls back the transaction. You must then increase the `Segment Size` property to an amount larger than the message.

A high segment size setting can be advantageous in that cleanup runs less often, although each cleanup takes somewhat longer. However, cleaning up two small segments still requires more time than cleaning up one large segment. Therefore, a large segment size can increase performance on systems that are constrained by disk I/O speed rather than memory or storage space.

Allowed Values

An integer greater than **1**. Set this property to at least twice the total number of anticipated durable subscribers.

Default Value

The default segment size is `16,384` pages, which is 8 MBytes for Windows and 16 MBytes for UNIX.

Note – Modifying the Segment Size requires changes to the database files. You must manually delete the existing database files and restart the domain server before the changes actually occur. To prevent message loss, make sure that no unread messages remain before deleting the database files.

Minimum Number of Segments

The `Minimum Number of Segments` property specifies the minimum number of database files (segments) that the JMS IQ Manager creates initially for stable message storage. When the minimum is exceeded, the server allocates additional segments on an as-needed basis, up to the number of files specified for the `Maximum Number of Segments` property as described in "Maximum Number of Segments" on page 96.

In addition to limiting the maximum number of segments, you can also specify the size limit for segments. For more information about the `Segment Size` property, refer to "Segment Size" on page 94.

Allowed Values

An integer from `1` through `99,999` indicating the number of segments.

Default Value

The default is `4` segments.

Maximum Number of Segments

The `Maximum Number of Segments` property specifies the upper limit for the number of database files (segments) that the JMS IQ Manager creates for its stable message storage. You use this property to limit the amount of disk space that the JMS IQ Manager uses. If the JMS IQ Manager attempts to write data that exceeds this limit, it exits gracefully and logs an error message in the JMS IQ Manager log.

Allowed Values

An integer from `0` through `99,999` indicating the number of segments.

Default Value

The default is `0`. This value causes the JMS IQ Manager to create new files as needed, limited only by available disk space.

Sync to Disk

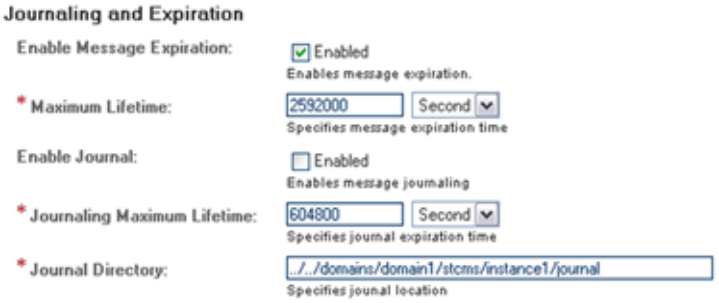
The `Sync to Disk` property specifies whether the JMS IQ Manager controls cache synchronization to disk. When you disable cache control, the operating system controls the synchronization schedule. Disabling cache control increases performance, but also increases risk of message loss in the event of system failure.

Default Condition

This property is *disabled* by default.

Journaling and Expiration Properties

FIGURE 18 Journaling and Expiration Properties Panel



Journaling messages enables you to republish messages at a later date. You can specify several options and properties for journaling, as described in the following sections:

To republish journaled messages, you use the STC MS Control utility as described in Example 19 or Example 20. You can also use the STC MS Control utility to browse journaled messages with the `-journaler` flag. For information, refer to Example 9.

Enable Message Expiration

Use the `Enable Message Expiration` option to enable or disable message expiration for JMS IQ Managers. When you enable message expiration, messages are removed from the queue after the time specified for the `Maximum Lifetime` property has expired.

Maximum Lifetime

The **Maximum Lifetime** property specifies the maximum amount of time before a live message expires. After it expires, the message is removed from the queue regardless of whether it has been consumed. If you specify `0`, the message never expires.

Default Value

The default is `2592000` seconds (30 days).

Enable Journal

Use the `Enable Journal` option to enable or disable journaling for JMS IQ Managers. When you enable journaling, every inbound message is automatically copied to the journal database. The message is then held in the journal database for the duration of the `time to live` value specified for journaled messages.

By default, the expiration time for a journaled message is 7 days. To change the time to live for journaled messages, refer to "Journaling Maximum Lifetime" on page 98. The `time to live` property for journaled messages is completely independent of when the live counterpart of the message is consumed by its publisher.

When a journaled message expires, it is not deleted from the journal database, but remains there until you back up the topics or queues. When you back up, all messages in the journal database are included in the archive, and the journal expired messages are removed from the journal database. You should back up daily when journaling is enabled; otherwise, the journal database retains journal expired messages and can grow exceedingly large. Because the journal database and the JMS IQ Manager database are located on the same system, you must guard against running out of disk space.

To back up the journal database, you use the MS Control Utility. For more information, refer to "To Create a Backup Archive File" on page 112.

When messages are in the journal database, you can view them but not edit them. You can use either Enterprise Manager or the STC MS Control utility to view and republish journaled messages. For more information about the MS Control utility, see "JMS Provider Management" on page 107.

Default Condition

Journaling is *disabled* by default.

Journaling Maximum Lifetime

The `Journaling Maximum Lifetime` property specifies the maximum amount of time that a journaled message persists before it expires. The JMS IQ Manager journals messages only when journaling is enabled as described in "Enable Journal" on page 97. Journaling is disabled by default.

When a journaled message expires, it is not deleted from the journal database, but remains there until you back up the topics or queues. When you back up, all messages in the journal database are included in the archive (`.zip`) file, and the expired messages are removed from the journal database.

Default Value

The default is `604800` seconds (7 days).

Journal Directory

The `Journal` directory holds the journal database files and the journaling log file. You can enter an absolute path or a path relative to the `..\logicalhost\is\domains` directory. Using an absolute path for the data directory enables you to store the journal database files on a different system, for example, for backup purposes.

The JMS IQ Manager creates a `journal` directory only when journaling is enabled. Journaling is *disabled* by default. For more information, refer to "Enable Journal" on page 97.

Allowed Values

An absolute path or a path relative to the `..\logicalhost\is\domains` directory.

Default Value

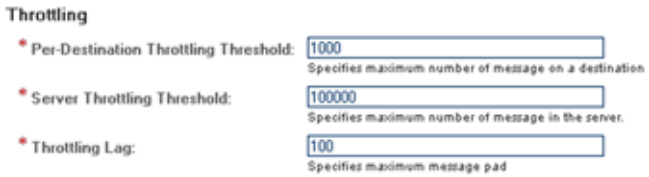
By default, the journal database files are stored in the following folder:`..\logicalhost\is\domains\domain_1\stcms\instance_1\journal`

Messaging Behavior Page

You can set the following properties on the Messaging Behavior page:

Throttling Properties

FIGURE 19 Throttling Properties Panel



As described in "Performance Issues" on page 32, you can use the JMS IQ Manager as a semi-permanent storage medium only if you have sufficient memory and disk resources. To manage the memory and disk resources needed by the JMS IQ Manager, you use the publisher throttling feature. You can specify several properties for throttling, as described in the following sections:

Per-Destination Throttling Threshold

The `Per-Destination Throttling Threshold` property specifies the maximum number of messages per topic or queue after which all producers of the message destination are throttled, assuming the `Server Throttling Threshold` has been exceeded. Once a producer is throttled, the JMS IQ Manager stops reading messages from it until the number of undelivered messages in the affected destination has dropped to less than the difference between the value specified by the `Per-Destination Throttling Threshold` and the value specified by the `Throttling Lag`. See also "Throttling Lag" on page 100.

Allowed Values

An integer from `0` through `999,999,999` indicating the number of messages. If you specify `0`, the publishers are never throttled.

Default Value

The default is `1000` messages.

Server Throttling Threshold

The `Server Throttling Threshold` property specifies the total number of messages for all message destinations combined before the JMS IQ Manager starts throttling any producers. For a detailed explanation and an example, see "Throttling Producers" on page 33.

Allowed Values

An integer from `0` through `999,999,999` indicating the number of messages. If you specify `0`, no producers are ever throttled.

Default Value

The default is `100,000` messages.

Throttling Lag

Use the `Throttling Lag` property in combination with the `Per-Destination Throttling Threshold` property. Once a producer is throttled, the JMS IQ Manager stops reading messages from it until the number of undelivered messages in the affected destination has dropped to less than the difference between the value specified by the `Per-Destination Throttling Threshold` and the value specified by the `Throttling Lag`. See also "Per-Destination Throttling Threshold" on page 99.

Allowed Values

An integer from `0` through `99,999,999`. The value must be less than that of the `Per-Destination Throttling Threshold` property.

Default Value

The default is `100` messages.

Special FIFO Mode Properties

FIGURE 20 Special FIFO Modes Properties Panel

As described in "Message Delivery Order" on page 15, the JMS IQ Manager allows three different FIFO delivery modes: fully concurrent, protected concurrent, or fully serialized. By default, all message destinations use the *fully concurrent* delivery mode. For more information, see Figure 5.

In fully concurrent mode, receivers can retrieve messages from a destination only when all other messages have been received or are in the process of being received. Receivers can then commit messages without restrictions. As a result, messages can be committed out of sequence, which is not always desirable.

In the `Special FIFO Modes` panel, you can specify alternate properties for delivery order, as described in the following sections:

- "Fully Serialized Queues" on page 101
- "Protected Concurrent Queues" on page 101
- "FIFO Expiration Time" on page 102

Fully Serialized Queues

The `Fully Serialized Queues` property specifies which queues are fully serialized. In fully serialized mode, receivers can only retrieve messages after all previous messages for the message destination have been received *and* committed.

To implement serialized mode across multiple application servers, you must set the JMS clients for the consumers involved to serial mode. For more information, refer to "Concurrency" on page 83.

Allowed Values

A list of existing queue names.

Protected Concurrent Queues

The `Fully Concurrent Queues` property specifies which queues use protected concurrent FIFO delivery mode. In protected concurrent mode, a receiver can retrieve messages just as in fully concurrent mode (only after all messages have been received or are being received), but messages can only be committed if all previous messages have been committed.

Allowed Values

A list of existing queue names.

FIFO Expiration Time

The `FIFO Expiration Time` property specifies the maximum amount of time to delay a *commit* request in special FIFO mode.

Default Value

The default FIFO mode expiration time is `604800` seconds (168 hours).

Time Dependency Properties

FIGURE 21 Time Dependency Properties Panel

By default, messages are processed by and delivered to Collaborations in the order in which they were committed to their destination, independent of messages associated with any other destination. Setting a specific time dependency causes the message processing order to be dependent on messages associated with *other* destinations. These destinations are specified using the `Time Dependency Topics` and `Time Dependency Queues` properties.

Messages associated with any of the destinations in the time dependency group are ordered in fully serialized mode. In other words, a message associated with a destination in this group is processed only after all older messages associated with any other destination in the time dependency group have been processed.

Note – Message properties such as JMS `priority` have no effect when time dependency is used.

For a general overview of message processing order, refer to "Message Delivery Order" on page 15.

Allowed Values

A list of queue or topic names.

If you specify a message destination that does not exist, Enterprise Manager enables time-based order for all other destinations and ignores the unknown name. This feature enables you to add topics and queues in your project at a later time.

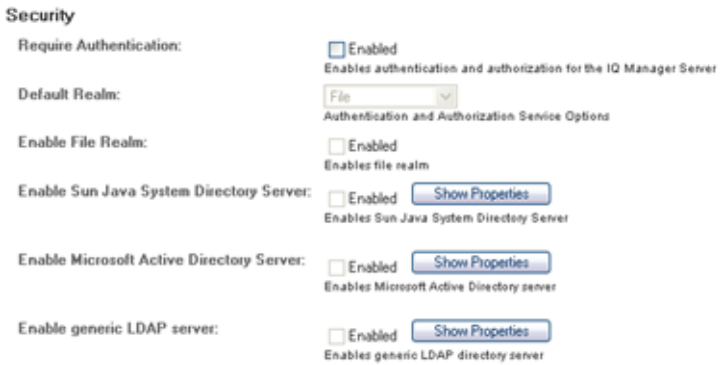
Note – When you specify time dependency, you cannot use a colon (:) or semicolon (;) in topic or queue names because they are already used for the time dependency value.

Access Control Page

Options on the `Access Control` page enable you to enable password authentication for the JMS IQ Manager and supported LDAP servers. Use of this page is described in *Using Enterprise Manager Management Application in Java CAPS.*

Security Options

FIGURE 22 Security Properties Panel



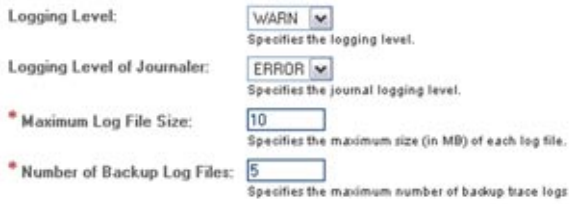
Default Condition

JMS IQ Manager security is *enabled* by default. To disable security, refer to *Using LDAP with Java CAPS*.

Diagnostics Page

Diagnostic Properties

FIGURE 23 Diagnostic Properties Panel



The generation of log files for diagnostic purposes can have a major impact on system resources. You can specify several properties for logging, as described in the following sections:

- "Logging Level" on page 104
- "Logging Level of Journaler" on page 105
- "Maximum Log File Size" on page 105
- "Number of Backup Log Files" on page 105

Logging Level

The `Logging Level` property specifies the threshold severity level at which the system issues log messages. The JMS IQ Manager only issues log messages having a severity level that is higher than or equal to the specified level.

TABLE 11 Logging Levels

Level	Message Types Issued
FATAL	Fatal
ERROR	Fatal, Error
WARN	Fatal, Error, Warning
INFO	All

Default Setting

By default, the logging level is specified as `WARN`, and the JMS IQ Manager issues warning, error, and fatal messages.

Logging Level of Journaler

The `Logging Level of Journaler` property specifies the threshold severity level at which the system journals log messages. The JMS IQ Manager only journals log messages having a severity level that is higher than or equal to the specified level.

TABLE 12 Journaler Logging Levels

Level	Message Types Journaled
FATAL	Fatal
ERROR	Fatal, Error
WARN	Fatal, Error, Warning
INFO	All

By default, the journal log file resides in the `journal` directory in the JMS IQ Manager directory. The location of this directory can be specified with the `Journal Directory` property as described in "Journal Directory" on page 98.

Default Condition

By default, the journaling level is specified as `ERROR`, and the JMS IQ Manager journal log includes only error and fatal messages.

Maximum Log File Size

You can specify the maximum size for the JMS IQ Manager log file with the `Maximum Log File Size` property. If the JMS IQ Manager attempts to log more than the specified log file size, the log file is renamed to `stcms.log.`*N*, and a new file is created.

The variable *N* is a number between 1 and the number specified by the `Number of Backup Log Files` property. By default, the JMS IQ Manager can create five backup log files.

Allowed Values

An integer larger than `0`, indicating the size of the log file in MBytes.

Default value

The default log file size is `10` MBytes.

Number of Backup Log Files

You can specify the maximum number of backup log files in the JMS IQ Manager database with the `Number of Backup Log Files` property. When the specified number is exceeded, the oldest log file is discarded.

Allowed Values

An integer greater than `0`, indicating the number of backup log files.

Default value

The default number of backup log files is `5`.

Miscellaneous Page

You can set the following property on the Miscellaneous page:

- "Enable Alert Option" on page 107

Enable Alert Option

FIGURE 24 Enable Alert Option Panel

When the `Enable Alert` option is enabled, the JMS IQ Manager will generate alerts when particular events occur, such as reaching a throttling threshold.

Default Condition

Alert generation is *disabled* by default.

JMS Provider Management

You can manage the JMS Provider either from Enterprise Manager or the command-line MS Control utility. This topic describes the use of the MS Control utility for managing JMS IQ Managers, Message Destinations, and the messages themselves. For information about Enterprise Manager, see *Using Enterprise Manager Management Application in Java CAPS*

Overview of MS Control Utility Features

The MS Control utility is a command-line utility that enables you to manage many advanced aspects of the JMS IQ Manager.

Message Servers

- Display the version of the JMS IQ Manager.
- Shut down the JMS IQ Manager.

Message Destinations

- For a specified JMS IQ Manager: List, create, or delete topics or queues.
- For a specified topic: List, create, or delete subscribers; retrieve a message list; view statistics.
- For a specified queue: List, create, or delete receivers; retrieve a message list; view queue statistics.

- For a specified queue or topic: Create, delete, modify, monitor, or list the contents of the queue or topic.

Messages

- For a specified message: View, delete, or modify message content.
- View or modify a particular message type.
- Fetch or delete a range of messages.
- Journal, back up, and archive messages.

MS Control Utility Details

The following links take you to detailed descriptions of the listed subtopics.

- "Flags and Arguments" on page 108
- "Using the MS Control Utility" on page 112
- "Command/Response Examples" on page 114

Flags and Arguments

This topic contains descriptions of the various flags and arguments used in the MS Control Utility.

TABLE 13 MS Control Utility Flags and Arguments

Flag *arguments*	Action and Comments	Shortcut
`-archive` *directoryname*	Displays the contents of the archive that has been extracted to the specified directory. The `-archive` flag displays the same information as does the `-journaler` flag, except that you specify the directory to which you extracted the archive. The MS Control utility displays information in the same format as `-journaler`. See "To Access an Archive File" on page 113.	`-ar`
`-backup` *file date*	Creates an archive (`.zip`) file that contains all messages for all queues and topics up to the specified date. Both live and journaled messages are included in the archive. See "To Create a Backup Archive File" on page 112.	`-b`
`-changeqmsg` *queuename seqnumber*	Changes the content of the message designated by the specified sequence number in the specified queue, reading from standard input (the command prompt, or whatever file or piped command it specifies). See "To Change Message Contents" on page 112.	`-cqm`

TABLE 13 MS Control Utility Flags and Arguments *(Continued)*

Flag *arguments*	Action and Comments	Shortcut
`-changetmsg` *topicname seqnumber*	Changes the content of the message designated by the specified sequence number in the specified topic, reading from standard input (the command prompt, or whatever file or piped command it specifies). See "To Change Message Contents" on page 112.	`-ctm`
`-createqueue` *queuename*	Creates a new queue with the specified name.	`-cq`
`-createsub` *topicname subname clientname*	Creates a new subscriber for the specified topic and client. For the name of the client, specify `eGate`.	`-cs`
`-createtopic` *topicname*	Creates a new topic with the specified name.	`-ct`
`-deletequeue` *queuename*	Deletes the specified queue.	`-dq`
`-deletesub` *topicname subname clientname*	Deletes the specified subscriber from the specified topic and client. For the name of the client, specify `eGate`.	`-ds`
`-deletetopic` *topicname*	Deletes the specified topic.	`-dt`
`-delqmsg` *queuename seqnumber*	Deletes the message designated by the specified sequence number in the specified queue.	`-dqm`
`-deltmsg` *topicname seqnumber*	Deletes the message designated by the specified sequence number in the specified topic.	`-dtm`
`--help`	Displays information about the JMS IQ Manager.	
`-host` *hostname*	Specifies the name of the Logical Host. If not specified, the default is `localhost`. For hosts other than `localhost` and flags other than `--help` and `--version`, specification of the host is required.	
`-journaler`	Displays the contents of journaled messages. Note that the journaler does not support information about subscribers and receivers. Flags such as `-sublistall`, `-deletesub`, or `-recvlistall` do not work with the `-journaler` flag. You cannot delete journaled messages, topics, or queues. The MS Control utility displays information about journaled topics and queues in the same format as the JMS IQ Manager	`-j`
`-locktopic` *topicname*	Locks a topic, preventing any subscriber from receiving messages from it.	`-lt`
`-msgtype` *type*	Specifies the data type of the content of the message, which must be either `bytes` or `text`.	
`-msversion`	Displays server version information.	

TABLE 13 MS Control Utility Flags and Arguments *(Continued)*

Flag *arguments*	Action and Comments	Shortcut
`-offset` *portoffset*	Specifies a server port offset number.	
`-port` *portnumber*	Specifies the TCP/IP port of the Logical Host to which this message server is listening. If not specified, the default is `7555`. For ports other than `7555` and flags other than `--help` and `--version`, specification of the port is required.	`-p`
`-qmessage` *queuename*	Retrieves the message designated by the specified sequence number for the specified queue. If the specified queue contains no message with this sequence number, an error is returned.	`-gqm`
`-qmimport` *queuename seqnumber numbermessages*	Republishes the specified number of messages from the specified queue, beginning with the specified sequence number.	`-qmi`
`-qmsglist` *queuename seqnumber numbermessages*	Lists all messages for the specified queue, starting at or above the specified sequence number and listing no more than the specified number of messages.	`-qml`
`-queuelist`	Lists all queues for this server.	`-ql`
`-queuestat` *queuename*	Displays the status of the specific queue.	`-qs`
`-recvlistall`	Lists all receivers for all queues for this server.	`-rla`
`-recvlistforqueue` *queuename*	Lists all receivers for the specified queue.	`-rlfq`
`-shutdown`	Shuts the server down. See "To Shut the Server Down" on page 114.	
`-status`	Displays the status of the server.	
`-sublistall`	Lists all subscribers to all topics for this server.	`-sla`
`-sublistfortopic` *topicname*	Lists all subscribers for the specified topic.	`-slft`
`-timeout` *seconds*	Specifies the timeout period in seconds (the default is `5` seconds). See "To Set the MS Control Utility Timeout Period" on page 113.	
`-tmessage` *topicname seqnumber*	Retrieves the message designated by the specified sequence number for the specified topic. If the specified topic contains no message with this sequence number, an error is returned.	`-gtm`
`-tmimport` *topicname seqnumber numbermessages*	Republishes the specified number of messages from the specified topic, beginning with the specified sequence number.	`-tmi`

TABLE 13 MS Control Utility Flags and Arguments *(Continued)*

Flag *arguments*	**Action and Comments**	**Shortcut**
`-tmsglist` *topicname seqnumber numbermessages*	Lists all messages for the specified topic, starting at or above the specified sequence number and listing no more than the specified number of messages.	`-tml`
`-topiclist`	List all topics for this server.	`-tl`
`-topicstat` *topicname*	Displays the status of the specified topic.	`-ts`
`-unlocktopic` *topicname*	Unlocks the specified topic, restoring access to all subscribers.	`-ut`
`-username` *username*	Specifies the user name to connect to the JMS IQ Manager. This flag is *mandatory* when JMS IQ security is enabled.	
`-userpassword` *userpassword*	Specifies the password to connect to the JMS IQ Manager. This flag is *mandatory* when JMS IQ security is enabled. Entering the password in the command line displays the password in plain text. To hide the text, enter an asterisk (*) for this option. The utility will prompt you for the password and the text will be hidden when you enter it.	
`--version`	Displays MS Utility version information.	

Syntax

The MS Control utility uses the following syntax for all flags other than `--help` and `--version`:

```
stcmsctrlutil -host hostname -port portnumber [-offset portoffset]
 -flag [argument1 ... argumentN]
```

If JMS IQ Manager security is enabled, you must also specify a user name and password as follows:

```
stcmsctrlutil -host hostname -port portnumber -username username
 -userpassword userpassword [-offset portoffset] -flag
```

For `--help` and `--version`, the syntax is as follows:

```
stcmsctrlutil --help
stcmsctrlutil --version
```

Using the MS Control Utility

This topic contains procedures for performing selected tasks using the MS Control utility.

Note – If you are using a Logical Host/Domain with a name other than `localhost`, you must also specify the host name by including the `-host` flag with the host name. Likewise, if you are using a non-default port for the JMS IQ Manager, you must also specify the port by including the `-port` flag with the port number.

▼ To Change Message Contents

To change the contents (payload) of a message for a specified message destination, use the `-changeqmsg` (`-cqm`) or `-changetmsg` (`-ctm`) flag. You must specify whether the original message type is bytes or text. The example illustrates the procedure for a topic.

Note – The message cannot be processed while you are changing the contents.

---

1. **Use the following command syntax to specify the contents change:**

   ```
   stcmsctrlutil.exe  -ctm topicname seqnumber -msgtype type
   ```

   where *topicname* is the name of the topic that contains the message, *seqnumber* is the sequence number of the message, and *type* is either `bytes` or `text` indicating the message type of the original message.

2. **Press Return.**

3. **Type the new content.**

4. **Press Control-Z.**

▼ To Create a Backup Archive File

This example creates an archive file containing all messages up to the date 09/11/2003 in the root backup path.

Note – See "Journal Directory" on page 98 regarding the location of the journal database directory.

Before You Begin Ensure that Journaling is enabled. For information, see "Enable Journal" on page 97.

1. **Use the following command syntax to specify the archive file:**

   ```
   stcmsctrlutil -backup journal-database-directory\backup\backup.zip "09/11/2003"
   ```

2. **Press Return.**

   If successful, you will receive the following response:

   ```
   Backup finished. Archived messages: 2003
   ```

▼ To Access an Archive File

Note – See "Journal Directory" on page 98 regarding the location of the journal database directory.

1. **Locate the desired archive file in the root backup directory.**

   By default, the root backup directory is:

   journal-database-directory`\backup`

2. **Extract the archive file to a directory of your own choosing.**

   For example:

   ```
   c:\JavaCAPS\backup\extract1
   ```

Example 1 Displaying a Topic List

As an example, use the following command syntax to display the topic list for the archive file:

```
stcmsctrlutil -ar c:\JavaCAPS\backup\extract1 -tl
```

You can then examine the individual topics further, as desired.

▼ To Set the MS Control Utility Timeout Period

The MS Control Utility uses a default timeout period of five seconds for retrieving requested information, after which the utility exits and you see the message "Timeout to receive message from the server, exiting stcmsctrlutil API." If this behavior should become a problem, you can include the `-timeout` flag with any command to set the MS Control utility's timeout period to a different value for that command only. Any command not including the `-timeout` flag uses the default value.

The example below sets the timeout period to 15 seconds, while obtaining a topic list from the journaler.

- **Use the following command syntax to set the timeout period:**

  ```
  stcmsctrlutil -j -tl -timeout 15
  ```

▼ To Shut the Server Down

- **Use the following command syntax to shut the server down:**

```
stcmsctrlutil  -shutdown
```

Command/Response Examples

This topic contains selected examples of commands and responses using typical values and data.

Message Server Example

EXAMPLE 2 Displaying JMS IQ Manager Status

Command:

```
stcmsctrlutil -host localhost -port 24055 -status
```

Response:

```
Up since: Tue Oct 14 20:54:23 2003
Memory used by data messages: 950.729 K(Bytes)
Total messages passed through: 1900331
Total messages retained: 3555
Number of message queue(s): 9
Number of connection(s): 14
Port number: 18007
Process ID: 2780
Server state: Ready and running...
```

Message Destination (Queue) Examples

EXAMPLE 3 Displaying an Active Queue List

Command:

```
stcmsctrlutil -host localhost -port 24055 -queuelist
```

Response:

```
Queue List:
    MyQueue0
    PTP
```

EXAMPLE 4 Displaying Active Queue Status

This example displays the current status of the queue PTP.

Command:

```
stcmsctrlutil -host localhost -port 24055 -queuestat PTP
```

Response:

```
Queue Name: PTP
First enqueue time: 02011970:00:00:00
Last enqueue time: 02011970:00:00:00
Number of current receivers: 2
Message count: 4
Messages sent and committed: 245
Min sequence number: 245
Max sequence number: 248
Suspended: No
```

The enqueue times and sequence numbers are described in "Enqueued Message Properties" on page 30.

EXAMPLE 5 Displaying Journaled Queue Status

This example displays journaled information about queue Q0.

Command:

```
stcmsctrlutil -j -qs Q0
```

Response:

```
Queue Name: Q0
First enqueue time: 09122003:00:14:07
Last enqueue time: 09122003:00:14:28
Number of current receivers: 0
Message count: 1001
Messages sent and committed: 0
Min sequence Number: 0
Max sequence Number: 1000
```

EXAMPLE 6 Displaying Properties of All Receivers

Command:

```
stcmsctrlutil -host localhost -port 24055 -recvlistall
```

Response:

```
Number Of Receiver(s): 3
Receiver ID: 14235659
        Queue name: MyQueue0
```

EXAMPLE 6 Displaying Properties of All Receivers *(Continued)*

```
        Session ID: 1
        Committed messages: 0
        Uncommitted messages: 0
Receiver ID: 14274653
        Queue name: PTP
        Session ID: 3
        Committed messages: 434
        Uncommitted messages: 0
Receiver ID: 14291939
        Queue name: PTP
        Session ID: 4
        Committed messages: 432
        Uncommitted messages: 1
```

EXAMPLE 7 Displaying Properties of All Receivers of Queues

Command:

```
stcmsctrlutil -host localhost -port 24055 -recvlistforqueue PTP
```

Response:

```
Number Of Receiver(s): 2
Receiver ID: 14274653
        Queue name: PTP
        Session ID: 3
        Committed messages: 434
        Uncommitted messages: 0
Receiver ID: 14291939
        Queue name: PTP
        Session ID: 4
        Committed messages: 432
        Uncommitted messages: 1
```

Message Destination (Topic) Examples

EXAMPLE 8 Displaying an Active Topic List

Command:

```
stcmsctrlutil -host localhost -port 24055 -topiclist
```

Response:

```
Topic List:
    SeeBeyond.MS.Control
    Broadcast
    STCTemporaryTopic.2.1
```

EXAMPLE 9 Displaying a Journaled Topic List

Command:

```
stcmsctrlutil -j -tl
```

Response:

```
Number Of Topic(s): 1
Topic List:
        T0
```

EXAMPLE 10 Displaying an Archived Topic List

This example displays the topic list for the archive file that has been extracted to the specified directory.

Command:

`stcmsctrlutil -ar` *archive-file-extract-directory* `-tl`

Response:

```
Number Of Topic(s): 1
Topic List:
        T0
```

EXAMPLE 11 Displaying Active Topic Status

This example displays the current status of the topic `Broadcast`.

Command:

```
stcmsctrlutil -host localhost -port 18007 -topicstat Broadcast
```

Response:

```
Topic Name: Broadcast
First enqueue time: 05172001:16:30:30
Last enqueue time: 05172001:16:30:42
Number of current subscribers: 0
Number of total subscribers: 2
Message count: 6
Min Sequence Number: 0
Max Sequence Number: 3
Suspended: No
```

The `Suspended` entry shows whether topic is suspended and all subscribers stop receiving messages. The status usually is `No`. After the JMS IQ Manager restarts, all topics show `Suspended: No status`.

The enqueue times and sequence numbers are described in "Enqueued Message Properties" on page 30.

EXAMPLE 12 Displaying Archived Topic Status

This example displays information about topic `T0`, in the archive file that has been extracted to the specified directory.

Command:

```
stcmsctrlutil -ar archive-file-extract-directory -ts T0
```

Response:

```
Topic Name: T0
First sequence number: 0
Last sequence number: 1000
First enqueue time: 09122003:00:14:17
Last enqueue time: 09122003:00:14:00
Number of current subscribers: 0
Number of total subscribers: 0
Message count: 1001
Lowest subscriber sequence: 0
Highest subscriber sequence: 0
```

EXAMPLE 13 Displaying Properties of All Subscribers to All Topics

Command:

```
stcmsctrlutil -host localhost -port 24055 -sublistall
```

Response:

```
Number Of Subscriber(s): 4
Subscriber name: NonDurable1
        Client ID:
        Topic name: SeeBeyond.MS.Control
        Committed sequence: 0
        High sequence: 0
Subscriber name: subscriber1
        Client ID: Client
        Topic name: Broadcast
        Committed sequence: 0
        High sequence: 3
Subscriber name: subscriber2
        Client ID: Client
        Topic name: Broadcast
        Committed sequence: 3
        High sequence: 6
Subscriber name: NonDurable2
        Client ID:
        Topic name: STCTemporaryTopic.2.1
        Committed sequence: 0
        High sequence: 0
```

EXAMPLE 14 Displaying Properties of All Subscribers to a Specific Topic

Command:

```
stcmsctrlutil -host localhost -port 24055 -sublistfortopic STC
```

Response:

```
Number Of Subscriber(s): 2
Subscriber name: subscriber1
        Client ID: Client
        Topic name: STC
        Committed sequence: 0
        High sequence: 3
Subscriber name: subscriber2
        Client ID: Client
        Topic name: STC
        Committed sequence: 3
        High sequence: 6
```

Message Examples

EXAMPLE 15 Changing Message Contents in a Topic

The following command changes the contents of a message on topic `T0`.

```
stcmsctrlutil -ctm T0 182 -p 18007 -msgtype text
NEWCONTENTS^Z
```

Response:

```
Message: 182 has been changed
```

The following command displays the revised contents of the message.

```
stcmsctrlutil.exe -p 18007 -tmessage T0 182 -msgtype text
```

Response:

NEWCONTENTS

EXAMPLE 16 Displaying Active Message Properties

This example requests information about a specific message in queue `Q0` with sequence number `0`.

Command:

```
stcmsctrlutil -qml Q0 0 1
```

Response:

EXAMPLE 16 Displaying Active Message Properties *(Continued)*

```
Number Of Messages(s): 0
```

In this example, the message has been consumed. As a result, the information is no longer available from the JMS IQ Manager. You must obtain the information from a journal or an archive.

EXAMPLE 17 Displaying Journaled Message Properties

This example requests information about a specific message in queue `Q0` with sequence number 0 from the journaler. As long as the journal has not expired, the message properties are available for viewing.

Command:

```
stcmsctrlutil -j -qml Q0 0 1
```

Response:

```
Number Of Messages(s): 1
Message[1]:
Message.SeqNo=0
Message.Timestamp=1031789647260
Journaler.ExpirationTime=1031809647260
Message.Size=228
Message.JMSProperty.TS=1031789647260
Message.JMSProperty.EX=0
Message.JMSProperty.DM=1
Message.JMSProperty.TY=ASCII
Message.JMSProperty.PR=0
Message.JMSProperty.RD=false
Message.JMSProperty.MI=ID:377:3b742aa5:950:0a01beee:3d7fdc4f104
Message.UserProperty.JMS_ProducerID=BENCH
```

EXAMPLE 18 Displaying Archived Message Properties

This example displays a description of the message with sequence number 1, in topic `T0`, in the archive file that has been extracted to the specified directory.

Command:

```
stcmsctrlutil -ar archive-file-extract-directory -tml T0 1 1
```

Response:

```
Number Of Messages(s): 1
Message[1]:
Message.SeqNo=1
Message.Timestamp=1031789654330
Journaler.ExpirationTime=1031809654330
Message.Size=228
```

EXAMPLE 18 Displaying Archived Message Properties *(Continued)*

```
Message.JMSProperty.EX=0
Message.JMSProperty.TS=1031789654330
Message.JMSProperty.DM=1
Message.JMSProperty.TY=ASCII
Message.JMSProperty.PR=0
Message.JMSProperty.MI=ID:45c:3b742aa6:950:0a01beee:3d7fdc5614a
Message.JMSProperty.RD=false
Message.UserProperty.JMS_ProducerID=BENCH
```

EXAMPLE 19 Republishing Journaled Messages from Topics

This example republishes five journaled messages from topic `T0` starting from message with sequence number `491`.

Command:

```
stcmsctrlutil -j -tmi T0 491 5
```

Response:

```
Executed function: IMPORT
Importing messages
Last imported sequence number = 491
Last imported sequence number = 497
```

EXAMPLE 20 Republishing Journaled Messages from Queues

This example republishes five journaled messages from queue `T0` starting from message with sequence number `500`.

Command:

```
stcmsctrlutil -j -qmi T0 500 5
```

Response:

```
Executed function: IMPORT
Importing messages
Last imported sequence number = 500
Import failed
Import failed on sequence number: 500
```

Note – This example failed, because no messages exist past sequence number 500.

Troubleshooting

This topic contains descriptions of suggested solutions for problems you may encounter when using the JMS IQ Manager.

Timestamp Errors

If the timestamps in the `stcms.log` are not correct, given the system time zone as specified by the TZ environment variable, shut down the domain and edit the following file:

`logicalhost/is/domains/`*`domain-name`*`/config/domain.xml`

Insert the following line after the line `<!-- various required jvm-options -->`:

`<jvm-options>-Dstcms.timezone=`*`your-timezone`*`</jvm-options>`

EXAMPLE 21 Timestamp Correction

```
<!-- various required jvm-options -->
<jvm-options>-Dstcms.timezone=PST8PDT</jvm-options>
```

Index

T

X