JBoss Enterprise Application Platform Common Criteria Certification 5

JBoss Messaging User Guide

for use with JBoss Enterprise Application Platform 5 Common Criteria Certification

Edition 5.1.0

Last Updated: 2017-11-20

JBoss Enterprise Application Platform Common Criteria Certification 5

JBoss Messaging User Guide

for use with JBoss Enterprise Application Platform 5 Common Criteria Certification
Edition 5.1.0

Tim Fox

Jeff Mesnil

Andy Taylor

Clebert Suconic

Howard Gao

Edited by

Laura Bailey

Jared Morgan

Abstract

A guide to using JBoss Messaging 1.4 with the JBoss Enterprise Application Platform 5.1.0.

Table of Contents

CHAPTER 1. ABOUT JBOSS MESSAGING 1.4

JBoss Messaging is an enterprise messaging system from JBoss. It is a complete rewrite of JBossMQ, the legacy JBoss Java Message Service (JMS) provider.

JBoss Messaging is the default JMS provider in JBoss Enterprise Application Platform 4.3 and 5.

JBoss Messaging is integral to Red Hat's messaging strategy. It offers improvements to performance in both single node and clustered environments, and features a modular architecture so that we can easily add more features in the future.

This guide shows you how to install, set up, and configure JBoss Messaging for JBoss Enterprise Application Platform.

CHAPTER 2. INTRODUCTION

JBoss Messaging provides an open-source and standards-based messaging platform to bring enterprise-class messaging to the mass market.

JBoss Messaging implements a robust, high-performance messaging core designed to support Service-Oriented Architectures (SOAs), Enterprise Service Buses (ESBs), and other integration requirements regardless of the level of demand.

JBoss Messaging lets you distribute application load evenly across your cluster. It balances each node's CPU cycles with no single point of failure, providing a highly scalable and performant clustering implementation.

JBoss Messaging includes a Java Messaging Service (JMS) front-end so that messages are delivered in a standards-based format, and to enable support for other messaging protocols in the future.

2.1. JBOSS MESSAGING FEATURES

JBoss Messaging provides the following features:

- A strong focus on performance, reliability and scalability with high throughput and low latency.
- A foundation for JBoss ESB for SOA initiatives. (JBoss ESB uses JBoss Messaging as its default JMS provider.)

JBoss Messaging also includes:

- publish-subscribe and point-to-point messaging models;
- persistent and non-persistent messages;
- guaranteed message delivery that ensures messages arrive once and only once where required;
- a transactional and reliable interface that supports ACID semantics;
- a customizable JAAS-based security framework;
- complete integration with JBoss Transactions (previously Arjuna JTA) to support full transaction recovery;
- an extensive JMX management interface;
- support for most major databases, including Oracle, DB2, Sybase, Microsoft SQL Server, PostgreSQL and MySQL;
- HTTP transport, for use with firewalls that allow only HTTP traffic;
- servlet transport to allow messaging through a dedicated servlet;
- SSL transport;
- configurable Dead Letter Queues (DLQs) and Expiry Queues;
- message statistics; which provide a rolling historical view of the messages delivered to queues and subscriptions;

- the automatic paging of messages to storage, which lets you use very large queues that would be too large to fit entirely within system memory; and
- strict message ordering which results in messages belonging to a particular message group being delivered according to the order of their arrival at the target queue.

JBoss Messaging also includes the following clustering features:

- **Fully-clustered queues and topics**

  Logical queues and topics are distributed across the cluster. You can send or receive a queue or topic to or from any node on the cluster.

- **Fully-clustered durable subscriptions**

  A particular durable subscription can be accessed from any node of the cluster, letting you spread processing load from that subscription across the entire cluster.

- **Fully-clustered temporary queues**

  If a sent message includes the `replyTo` of a temporary queue, it can be returned on any node of the cluster.

- **Intelligent message redistribution**

  Messages are automatically moved between nodes of the cluster to take advantage of different consumer speeds on different nodes. This helps to prevent starvation or build-up of messages on a particular node.

- **Message order protection**

  Enable this to ensure that the order of messages produced by a producer is identical to the order of messages consumed by a consumer. This works even if message redistribution is active.

- **Completely transparent failover**

  When a server fails, your sessions continue exception-free on a new node. This is also completely configurable: if you do not want to implement this failover behavior, you can disable it and fall back to exceptions being thrown and manually recreating connections on a new node.

- **High availability and seamless failover**

  If the node fails, you will automatically failover to a different node without losing any persistent messages and can seamlessly continue your session. *Once and only once*delivery of persistent messages is respected at all times.

- **Message bridge**

  JBoss Messaging contains a message bridge component, which lets you bridge messages between any two JMS 1.1 destinations. This lets you connect geographically separate clusters and form large, globally-distributed logical queues and topics.

2.2. COMPATIBILITY WITH JBOSS MQ

JBoss MQ was the JMS implementation shipped with Enterprise Application Platform 4.2. Since JBoss Messaging is compatible with both JMS 1.1 and JMS 1.0.2b, the JMS code written against JBoss MQ will run with JBoss Messaging without any further changes.

JBoss Messaging has no wire format compatibility with JBoss MQ. It is therefore necessary to upgrade JBoss MQ clients with JBoss Messaging client JARs.

IMPORTANT

Although JBoss Messaging deployment descriptors are similar to JBoss MQ deployment descriptors, they are *not identical*, and will require some simple adjustments before they will work with JBoss Messaging. The database data model is completely different, so JBoss Messaging should not be used with a JBoss MQ data schema, or vice-versa.

2.3. SYSTEM PROPERTIES USED BY JBOSS MESSAGING

2.3.1. support.bytesId

This system property controls the default behavior when constructing a `JBossMessage` object from a foreign message object. Set this property via the command line on server startup with the `-D` option.

If this property is set to `true`, the `JBossMessage` constructor will try to extract the native byte[] correlation ID from the foreign message headers. If set to `false`, it will use the normal string type `JMSCorrelationID`. This property will default to `true` if not set or when set to something other than `true` or `false`.

2.3.2. retain.oldxabehaviour

This system property controls the type of exception thrown by a JMS XAResource in the event that the `prepare()` method is called after the connection is broken. Set this property via the command line on server startup with the `-D` option.

If this property is not defined, an `XAException` with an `XA_RBCOMMFAIL` error code will be thrown. Otherwise an `XAException` with an `XA_RETRY` error code will be thrown. It should be noted that JBoss Messaging does not define this property by default.

CHAPTER 3. JBOSS MESSAGING INSTALLATION

JBoss Enterprise Application Platform (EAP) comes with JBoss Messaging pre-installed as the default JMS provider. If you are using EAP version 4.3 or higher, there is no need to manually install JBoss Messaging.

CHAPTER 4. RUNNING THE EXAMPLES

JBoss Messaging has a number of examples that are available for download. Refer to the *Installation Guide* shipped with JBoss Enterprise Application Platform to learn how to install and configure the examples. The *Installation Guide* is available from http://docs.redhat.com/ under *JBoss Enterprise Application Platform > 5.0*.

Unzip the examples file to the **`$JBOSS_HOME/docs/examples`** directory.

Before you run these examples, deploy the **`jbm-examples-destinations-service`** located under **`$JBOSS_HOME/docs/examples/jboss-messaging-examples/destinations/`** to **`$JBOSS_HOME/server/default/deploy`**. The

The following examples are included with JBoss Messaging:

`$JBOSS_HOME/docs/examples/jboss-messaging-examples/queue/`

This example shows a simple send and receive to a remote queue using a JMS client.

`$JBOSS_HOME/docs/examples/jboss-messaging-examples/queue-failover/`

This example demonstrates the transparent failover of a JMS consumer.

`$JBOSS_HOME/docs/examples/jboss-messaging-examples/topic/`

This example shows a simple send and receive to a remote topic using a JMS client.

`$JBOSS_HOME/docs/examples/jboss-messaging-examples/mdb/`

This example demonstrates the use of an Enterprise JavaBean 2.1 MDB with JBoss Messaging.

`$JBOSS_HOME/docs/examples/jboss-messaging-examples/ejb3mdb/`

This example demonstrates the use of an Enterprise JavaBean 3.0 MDB with JBoss Messaging.

`$JBOSS_HOME/docs/examples/jboss-messaging-examples/stateless/`

This example demonstrates an Enterprise JavaBean 2.1 stateless session bean interacting with JBoss Messaging.

`$JBOSS_HOME/docs/examples/jboss-messaging-examples/mdb-failure/`

This example demonstrates rollback and redelivery within an Enterprise JavaBean 2.1.

`$JBOSS_HOME/docs/examples/jboss-messaging-examples/secure-socket/`

This example demonstrates a JMS client interacting with a JBoss Messaging server via SSL-encrypted transport.

`$JBOSS_HOME/docs/examples/jboss-messaging-examples/http/`

This example demonstrates a JMS client interacting with a JBoss Messaging server by tunneling traffic via HTTP.

`$JBOSS_HOME/docs/examples/jboss-messaging-examples/web-service/`

This example demonstrates the JBoss Webservice interacting with JBoss Messaging.

`$JBOSS_HOME/docs/examples/jboss-messaging-examples/distributed-queue/`

This example demonstrates a JMS client interacting with a JBoss Messaging distributed queue. Two instances of JBoss Enterprise Application Platform are required to run this example.

`$JBOSS_HOME/docs/examples/jboss-messaging-examples/distributed-topic/`

This example demonstrates a JMS client interacting with a JBoss Messaging distributed topic. Two instances of JBoss Enterprise Application Platform are required to run this example.

`$JBOSS_HOME/docs/examples/jboss-messaging-examples/servlet/`

This example demonstrates how to use servlet transport with JBoss Messaging. It deploys a servlet and a `ConnectionFactory` which uses the servlet transport.

`$JBOSS_HOME/docs/examples/jboss-messaging-examples/ordering-group/`

This example demonstrates the use of strict message ordering. It uses the JBoss Messaging ordering group API to deliver strictly ordered messages, regardless of the message priority.

`$JBOSS_HOME/docs/examples/jboss-messaging-examples/bridge/`

This example demonstrates the use of a message bridge. It deploys a message bridge within JBoss Enterprise Application Platform, which moves messages from a source to a target queue.

Familiarize yourself with the examples to gain a thorough understanding of JBoss Messaging.

IMPORTANT

The non-clustered examples expect an instance of JBoss Enterprise Application Platform to be running with all default settings. The `readme.html` for each example provides the setup details, expected output, and simple troubleshooting.

IMPORTANT

The clustered examples require two instances of JBoss Enterprise Application Platform to be running, with correctly configured cluster names and port settings. The `readme.html` for each example provides the setup details, expected output, and simple troubleshooting.

CHAPTER 5. CONFIGURATION

The Java Message Service (JMS) API specifies how a messaging client interacts with a messaging server. How messaging services such as message destinations and connection factories are defined and implemented depends on the JMS provider. JBoss Messaging has its own files for service configuration.

This chapter shows you how to configure various services available in JBoss Messaging that work together to provide JMS API-level services to client applications.

JBoss Messaging configuration is divided between several configuration files. Depending on the type of service provided, configuration information is divided between **`messaging-service.xml`**, **`remoting-bisocket-service.xml`**, **`<your database type>-persistence-service.xml`**, **`connection-factories-service.xml`** and **`destinations-service.xml`**. These files can all be found in the **`$JBOSS_HOME/server/$PROFILE/deploy/messaging`** directory.

AOP interceptor stacks can be configured in **`aop-messaging-client.xml`** (for client-side behavior) and **`aop-messaging-server.xml`** (for server-side behavior). There is usually no need to change these files, but some interceptors can be removed to improve performance if they are not required. Ensure that you have considered the security implications before removing the security interceptor.

5.1. CONFIGURING THE SERVERPEER

The **`ServerPeer`** is the heart of the JBoss Messaging JMS facade. You can configure its behavior by altering **`$JBOSS_HOME/server/$PROFILE/deploy/messaging/messaging-service.xml`**.

All JBoss Messaging services are based in the **`ServerPeer`**.

An example of a Server Peer configuration is presented below. Note that not all values for the server peer's attributes are specified in the example

```
<!-- ServerPeer MBean configuration
    ============================== -->

<mbean code="org.jboss.jms.server.ServerPeer"
  name="jboss.messaging:service=ServerPeer"
  xmbean-dd="xmdesc/ServerPeer-xmbean.xml">

  <!--The unique id of the server peer - in a cluster each node
      MUST have a unique value - must be an integer-->

  <attribute name="ServerPeerID">
    ${jboss.messaging.ServerPeerID:0}
  </attribute>

  <!--The default JNDI context to use for queues when they are
      deployed without specifying one-->

  <attribute name="DefaultQueueJNDIContext">/queue</attribute>

  <!--The default JNDI context to use for topics when they are
      deployed without specifying one -->

  <attribute name="DefaultTopicJNDIContext">/topic</attribute>
```

```
<attribute name="PostOffice">
  jboss.messaging:service=PostOffice
</attribute>

<!-- The default Dead Letter Queue (DLQ) to use for destinations.
     This can be overridden on a per destinatin basis -->

<attribute name="DefaultDLQ">
  jboss.messaging.destination:service=Queue,name=DLQ
</attribute>

<!--The default maximum number of times to attempt delivery of a
    message before sending to the DLQ (if configured).
    This can be overridden on a per destination basis-->

<attribute name="DefaultMaxDeliveryAttempts">10</attribute>

<!--The default Expiry Queue to use for destinations. This can
    be overridden on a per destinatin basis-->

<attribute name="DefaultExpiryQueue">
  jboss.messaging.destination:service=Queue,name=ExpiryQueue
</attribute>

<!--The default redelivery delay to impose. This can be overridden
    on a per destination basis -->

<attribute name="DefaultRedeliveryDelay">0</attribute>

<!--The periodicity of the message counter manager enquiring on
    queues for statistics-->

<attribute name="MessageCounterSamplePeriod">5000</attribute>

<!--The maximum amount of time for a client to wait for failover
    to start on the server side after it has detected failure-->

<attribute name="FailoverStartTimeout">60000</attribute>

<!--The maximum amount of time for a client to wait for failover
    to complete on the server side after it has detected failure-->

<attribute name="FailoverCompleteTimeout">300000</attribute>

<attribute name="StrictTck">false</attribute>

<!--The maximum number of days results to maintain in the message
    counter history-->

<attribute name="DefaultMessageCounterHistoryDayLimit">-1</attribute>

<!--The name of the connection factory to use for creating
    connections between nodes to pull messages-->

<attribute name="ClusterPullConnectionFactoryName">
  jboss.messaging.connectionfactory:service=ClusterPullConnectionFactory
```

```
    </attribute>

    <!--When redistributing messages in the cluster. Do we need to
        preserve the order of messages received
        by a particular consumer from a particular producer? -->

    <attribute name="DefaultPreserveOrdering">false</attribute>

    <!-- Max. time to hold previously delivered messages back waiting for
         clients to reconnect after failover -->

    <attribute name="RecoverDeliveriesTimeout">300000</attribute>

    <!-- Set to true to enable message counters that can be viewed via JMX -
  ->

    <attribute name="EnableMessageCounters">false</attribute>

    <!-- The password used by the message sucker connections to create
  connections.
         THIS SHOULD ALWAYS BE CHANGED AT INSTALL TIME TO SECURE SYSTEM
    <attribute name="SuckerPassword"></attribute>
    -->

    <!-- The name of the server aspects configuration resource
    <attribute name="ServerAopConfig">aop/jboss-aop-messaging-
  server.xml</attribute>
    -->
    <!-- The name of the client aspects configuration resource
      <attribute name="ClientAopConfig">aop/jboss-aop-messaging-
  client.xml</attribute>
    -->

    <depends optional-attribute-name="PersistenceManager">
      jboss.messaging:service=PersistenceManager
    </depends>

    <depends optional-attribute-name="JMSUserManager">
      jboss.messaging:service=JMSUserManager
    </depends>

    <depends>jboss.messaging:service=Connector,transport=bisocket</depends>
    <depends optional-attribute-name="SecurityStore"
      proxy-type="org.jboss.jms.server.SecurityStore">
        jboss.messaging:service=SecurityStore
    </depends>
  </mbean>
```

5.2. SERVERPEER ATTRIBUTES

This section discusses the **ServerPeer** managed bean attributes.

ServerPeerID

The unique identifier of the `ServerPeer`. Each node deployed must have a unique identifier, whether the nodes form a cluster or are linked by a message bridge. The identifier must be a valid integer.

DefaultQueueJNDIContext

The default JNDI context to be used when binding queues. The default value is **`/queue`**.

DefaultTopicJNDIContext

The default JNDI context to be used when binding topics. The default value is **`/topic`**.

PostOffice

The post office used by the `ServerPeer`. You will not normally need to edit this attribute. The post office routes messages to queues and maintains the mapping between queues and addresses.

DefaultDLQ

The default DLQ (Dead Letter Queue) that the server uses for destinations. You can override the DLQ on a per-destination basis. For more information, see Section 5.7, “Configuring Destinations”. A DLQ is a destination for messages that the server has failed to deliver more than a certain number of times. If the DLQ is not specified, the message will be removed after the maximum number of delivery attempts. You can specify a global default for the maximum number of delivery attempts with the **`DefaultMaxDeliveryAttempts`** attribute, or set the maximum individually on a per-destination basis.

DefaultMaxDeliveryAttempts

The global default for the maximum number of times delivery will be attempted for a message before the message is removed or sent to the DLQ, if configured. The default value is **`10`**. You can override this value on a per-destination basis.

DefaultExpiryQueue

The default expiry queue that the `ServerPeer` will use for destinations. You can override this value on a per-destination basis, as seen in the section on destination managed bean configuration. An expiry queue holds messages that have expired. Message expiry is determined by the value of **`Message::getJMSExpiration()`**. If the expiry queue is not specified, the message will be deleted when it expires.

DefaultRedeliveryDelay

This attribute lets you delay a redelivery attempt, which helps to prevent thrashing delivery-failure. The default value is **`0`** (that is, no delay). You can override this value on a per-destination basis.

MessageCounterSamplePeriod

This attribute defines the period of time between the server's queries to the queue for queue statistics. The default value is **`5000`** milliseconds.

FailoverStartTimeout

The longest period (in milliseconds) that the client will wait for failover to begin on the server side when a problem is detected. The default value is **`60000`** (one minute).

FailoverCompleteTimeout

The longest period (in milliseconds) that the client will wait for failover to complete on the server side once failover has been initiated. The default value is `300000` (five minutes).

DefaultMessageCounterHistoryDayLimit

JBoss Messaging provides a message counter history, which shows the number of messages arriving on each queue over a certain number of days. This attribute represents the maximum number of days for which to store message counter history. You can override this value on a per-destination.

ClusterPullConnectionFactoryName

The connection factory used to pull, or suck, messages between queues. You can omit this attribute to disable message sucking while retaining failover.

DefaultPreserveOrdering

When `true`, JMS ordering is preserved in the cluster. See Chapter 6, *Clustering Notes* for more detail. The default value is `false`.

RecoverDeliveriesTimeout

When failover occurs, messages that have been delivered will be stored while the clients reconnect. If the clients do not reconnect (for example, if the client is dead), these messages will eventually time out and be added to the queue. This attribute sets the period before timeout in milliseconds. The default value is `300000` (five minutes).

EnableMessageCounters

When set to `true`, enables message counters upon server start.

SuckerPassword

JBoss Messaging internally creates connections between nodes to redistribute messages between clustered destinations. These connections are created with a special, reserved username. This attribute defines the password to use when creating these connections.

For versions of JBoss Messaging later than 1.4.1.GA, you must define the `SuckerPassword` on the `SecurityMetadataStore`.

WARNING

The `SuckerPassword` must be changed at install time, or the default password will be used, giving any user who knows the default password access to any destination on the server.

SuckerConnectionRetryTimes

This is the maximum number of times a sucker's connection is permitted to retry in the event of a failure. The default value is `-1` which represents "retry indefinitely".

SuckerConnectionRetryInterval

This is the interval in milliseconds between each retry of the failed sucker's connection. The default value is **`5000`**.

StrictTck

To enable strict JMS Technology Compatibility Kit (TCK) semantics, set this attribute to **`true`**.

Destinations

Returns a list of the destinations (queues and topics) currently deployed.

MessageCounters

A message counter for a particular queue.

MessageStatistics

Statistics about each message counter for each queue.

SupportsFailover

When this attribute is **`false`**, server-side failover does not occur when a node crashes in a cluster.

PersistenceManager

The persistence manager used by the **`ServerPeer`**. (You will not normally need to change this attribute.)

JMSUserManager

The JMS user manager used by the **`ServerPeer`**. (You will not normally need to change this attribute.)

SecurityStore

The pluggable **`SecurityStore`**. If you redefine this attribute, remember that you will need to authenticate the **`MessageSucker`** user (**`JBM.SUCKER`**) with all special permissions required by clustering.

SupportsTxAge

Specifies whether the transaction creation time is stored in the transaction record. If set to **`true`**, the transaction record is stored. The default is **`false`**.

5.2.1. ServerPeer methods

The following methods are available for the ServerPeer managed bean:

deployQueue

Used to programmatically deploy a queue. If the queue exists but is undeployed, it will be deployed. Otherwise, it is created and deployed.

The **`name`** parameter matches a destination to deploy.

The optional **`jndiName`** parameter represents the full JNDI name of the location to which a destination will be bound. If this is not specified, the destination will be bound in **`<DefaultQueueJNDIContext>/<name>`**.

There are two overloaded versions of this operation. The first deploys the destination with default paging parameters. The second deploys the destination with the paging parameters specified. For more information about paging parameters, see Section 5.7, “Configuring Destinations”.

undeployQueue

Used to programmatically undeploy a queue. Queues are not removed from persistent storage. This operation returns `true` if the queue is successfully undeployed. Otherwise, it returns `false`.

destroyQueue

Used to programmatically destroy a queue. Queues are undeployed and all of their data is removed from the database and destroyed.

> **WARNING**
>
> Exercise caution when using this method, since it will delete all data for the queue.

This operation returns `true` if the queue was destroyed successfully. Otherwise, it returns `false`.

deployTopic

Used to programmatically deploy a topic. There are two overloaded versions of this operation. The first deploys already existing topics with the default paging parameters. The second creates and deploys topics with specified paging parameters. See Section 5.7, “Configuring Destinations” for more information.

The `name` parameter represents the name of the destination to deploy.

The `jndiName` represents the full JNDI name of the location to which the destination will be bound. If this is not specified, the destination will be bound in `<DefaultTopicJNDIContext>/<name>`.

undeployTopic

Used to programmatically undeploy a topic. Topics are undeployed, but not removed from persistent storage. This operation returns `true` if the topic is undeployed successfully. Otherwise, `false` is returned.

destroyTopic

Used to programmatically destroy a topic. Topics are undeployed and all data is removed from the database and destroyed. This operation returns `true` if the topic is successfully destroyed. Otherwise, it returns `false`.

WARNING

Exercise caution when using this method: it will delete all data for the topic.

listMessageCountersHTML

Returns message counters in a simply-displayed HTML format.

resetAllMesageCounters

Resets all message counters to zero.

enableMessageCounters

Enables all message counters for all destinations. Message counters are disabled by default.

disableMessageCounters

Disables all message counters for all destinations. Message counters are disabled by default.

retrievePreparedTransactions

Retrieves a list of the XIDs for all transactions currently in a prepared state on the node.

showPreparedTransactions

Retrieves a list of the XIDs for all transactions currently in a prepared state on the node in an easily-displayed HTML format.

listAllPreparedTransactions

Displays the details of all prepared transactions.

listPreparedTransactions

Displays the details of all prepared transactions where the transaction ages are equal to or older than a specified time.

showMessageDetails

Displays the details of a message. The message ID is used to specify the message to display.

commitPreparedTransaction

Manually commit a prepared transaction. The transaction ID is used to specify the transaction to commit.

rollbackPreparedTransaction

Manually roll-back a prepared transaction. The transaction ID is used to specify the transaction to roll-back.

5.3. CHANGING THE DATABASE

WARNING

The default persistence configuration works with Hypersonic (HSQLDB) so that the JBoss Enterprise Platforms are able to run "out of the box". However, *Hypersonic is not supported in production and should not be used in a production environment.*

Known issues with the Hypersonic Database include:

- no transaction isolation
- thread and socket leaks (`connection.close()` does not tidy up resources)
- persistence quality (logs commonly become corrupted after a failure, preventing automatic recovery)
- database corruption
- stability under load (database processes cease when dealing with too much data)
- not viable in clustered environments

See the "Using Other Databases" chapter in the *Getting Started Guide* for further information.

The Persistence Manager, Post Office and JMS User Manager all interact with persistent storage. The Persistence Manager handles message-related persistence. The Post Office handles binding related persistence. The JMS User Manager handles user-related persistence. All configuration for these managed beans is handled in the `<your database type>-persistence-service.xml` file.

Example configuration files for MySQL, Oracle, PostgreSQL, Microsoft SQL Server or Sybase databases are available in the `jboss-as/docs/examples/jms` directory of the release bundle.

To enable support for one of these databases, replace the default `jboss-as/server/$PROFILE/deploy/messaging/hsqldb-persistence-service.xml` configuration file with the configuration file specific to your database type and restart the server.

By default, the messaging services relying on a data store reference `java:/DefaultDS` for the data source. To deploy a data source with a different JNDI name, you must update all `DataSource` attributes in the persistence configuration file. Example data source configurations are included in the distribution.

5.4. CONFIGURING THE POST OFFICE

The post office routes messages to their destination or destinations. It maintains the mappings between the addresses to which a message can be sent, and the final queue. For example, when sending a message with an address that represents a JMS queue, the post office routes the message to that JMS queue. When sending a message with an address that represents a JMS topic, the post office routes the message to a set of queues — one for each JMS subscription.

The post office also handles the persistence for address mapping.

JBoss Messaging post offices are cluster-aware. In a cluster, they automatically route (push) and pull messages between nodes in order to provide fully-distributed JMS queues and topics.

Configure the post office in the `<database type>-persistence-service.xml` file. For example:

```
<mbean code="org.jboss.messaging.core.jmx.MessagingPostOfficeService"
  name="jboss.messaging:service=PostOffice"
  xmbean-dd="xmdesc/MessagingPostOffice-xmbean.xml">

  <depends optional-attribute-name="ServerPeer">
    jboss.messaging:service=ServerPeer
  </depends>

  <depends>
    jboss.jca:service=DataSourceBinding,name=DefaultDS
  </depends>

  <depends optional-attribute-name="TransactionManager">
    jboss:service=TransactionManager
  </depends>

  <!-- The name of the post office -->

  <attribute name="PostOfficeName">JMS post office</attribute>

  <!-- The datasource used by the post office to access it's
       binding information -->

  <attribute name="DataSource">java:/DefaultDS</attribute>

  <!-- If true will attempt to create tables and indexes on
       every start-up -->

  <attribute name="CreateTablesOnStartup">true</attribute>

  <!-- If true then we will automatically detect and reject
       duplicate messages sent during failover -->

  <attribute name="DetectDuplicates">true</attribute>

  <!-- The size of the id cache to use when detecting duplicate
       messages -->

  <attribute name="IDCacheSize">500</attribute>

  <attribute name="SqlProperties">
    CREATE_POSTOFFICE_TABLE=CREATE TABLE JBM_POSTOFFICE
      (POSTOFFICE_NAME VARCHAR(255),
      NODE_ID INTEGER, QUEUE_NAME VARCHAR(255), COND VARCHAR(1023),
      SELECTOR VARCHAR(1023), CHANNEL_ID BIGINT, CLUSTERED CHAR(1),
      ALL_NODES CHAR(1),
      PRIMARY KEY(POSTOFFICE_NAME, NODE_ID, QUEUE_NAME)) ENGINE = INNODB

    INSERT_BINDING=INSERT INTO JBM_POSTOFFICE
      (POSTOFFICE_NAME, NODE_ID, QUEUE_NAME, COND, SELECTOR,
       CHANNEL_ID, CLUSTERED, ALL_NODES)
      VALUES (?, ?, ?, ?, ?, ?, ?, ?)

    DELETE_BINDING=DELETE FROM JBM_POSTOFFICE WHERE
      POSTOFFICE_NAME=? AND NODE_ID=? AND QUEUE_NAME=?

    LOAD_BINDINGS=SELECT QUEUE_NAME, COND, SELECTOR,
      CHANNEL_ID, CLUSTERED, ALL_NODES FROM
   JBM_POSTOFFICE WHERE POSTOFFICE_NAME=? AND NODE_ID=?
  </attribute>

  <!-- This post office is clustered. If you don't want a clustered post
       office then set to false -->

  <attribute name="Clustered">true</attribute>

  <!-- All the remaining properties only have to be specified if the post
       office is clustered. You can safely comment them out if your post
       office is non clustered -->

  <!-- The JGroups group name that the post office will use -->

  <attribute name="GroupName">
    ${jboss.messaging.groupname:MessagingPostOffice}
  </attribute>

  <!-- Max time to wait for state to arrive when the post office
       joins the cluster -->

  <attribute name="StateTimeout">5000</attribute>

  <!-- Max time to wait for a synchronous call to node members using
       the MessageDispatcher -->

  <attribute name="CastTimeout">50000</attribute>

  <!-- Set this to true if you want failover of connections to occur
       when a node is shut down -->

  <attribute name="FailoverOnNodeLeave">false</attribute>

  <!-- JGroups stack configuration for the data channel - used for sending
       data across the cluster -->

  <!-- By default we use the TCP stack for data -->
  <attribute name="DataChannelConfig">
    <config>
      <TCP start_port="7900"
        loopback="true"
        recv_buf_size="20000000"
        send_buf_size="640000"
        discard_incompatible_packets="true"
        max_bundle_size="64000"
        max_bundle_timeout="30"
        use_incoming_packet_handler="true"
```

```
          use_outgoing_packet_handler="false"
          down_thread="false" up_thread="false"
          enable_bundling="false"
          use_send_queues="false"
          sock_conn_timeout="300"
          skip_suspected_members="true"/>
        <MPING timeout="4000"
          bind_to_all_interfaces="true"
          mcast_addr="${jboss.messaging.datachanneludpaddress:228.6.6.6}"
          mcast_port="${jboss.messaging.datachanneludpport:45567}"
          ip_ttl="8"
          num_initial_members="2"
          num_ping_requests="1"/>
        <MERGE2 max_interval="100000"
          down_thread="false" up_thread="false" min_interval="20000"/>
        <FD_SOCK down_thread="false" up_thread="false"/>
        <VERIFY_SUSPECT timeout="1500" down_thread="false"
         up_thread="false"/>
        <pbcast.NAKACK max_xmit_size="60000"
          use_mcast_xmit="false" gc_lag="0"
          retransmit_timeout="300,600,1200,2400,4800"
          down_thread="false" up_thread="false"
          discard_delivered_msgs="true"/>
        <pbcast.STABLE stability_delay="1000" desired_avg_gossip="50000"
          down_thread="false" up_thread="false"
          max_bytes="400000"/>
        <pbcast.GMS print_local_addr="true" join_timeout="3000"
          down_thread="false" up_thread="false"
          join_retry_timeout="2000" shun="false"
          view_bundling="true"/>
      </config>
    </attribute>

    <!-- JGroups stack configuration to use for
         the control channel - used for control messages -->

    <!-- We use udp stack for the control channel -->
    <attribute name="ControlChannelConfig">
      <config>
        <UDP
          mcast_addr="${jboss.messaging.controlchanneludpaddress:228.7.7.7}"
          mcast_port="${jboss.messaging.controlchanneludpport:45568}"
          tos="8"
          ucast_recv_buf_size="20000000"
          ucast_send_buf_size="640000"
          mcast_recv_buf_size="25000000"
          mcast_send_buf_size="640000"
          loopback="false"
          discard_incompatible_packets="true"
          max_bundle_size="64000"
          max_bundle_timeout="30"
          use_incoming_packet_handler="true"
          use_outgoing_packet_handler="false"
          ip_ttl="2"
          down_thread="false" up_thread="false"
          enable_bundling="false"/>
        <PING timeout="2000"
          down_thread="false" up_thread="false" num_initial_members="3"/>
        <MERGE2 max_interval="100000"
          down_thread="false" up_thread="false" min_interval="20000"/>
        <FD_SOCK down_thread="false" up_thread="false"/>
        <FD timeout="10000" max_tries="5" down_thread="false"
          up_thread="false" shun="true"/>
        <VERIFY_SUSPECT timeout="1500" down_thread="false"
         up_thread="false"/>
        <pbcast.NAKACK max_xmit_size="60000"
          use_mcast_xmit="false" gc_lag="0"
          retransmit_timeout="300,600,1200,2400,4800"
          down_thread="false" up_thread="false"
          discard_delivered_msgs="true"/>
        <UNICAST timeout="300,600,1200,2400,3600"
          down_thread="false" up_thread="false"/>
        <pbcast.STABLE stability_delay="1000" desired_avg_gossip="50000"
          down_thread="false" up_thread="false"
          max_bytes="400000"/>
        <pbcast.GMS print_local_addr="true" join_timeout="3000"
          use_flush="true" flush_timeout="3000"
          down_thread="false" up_thread="false"
          join_retry_timeout="2000" shun="false"
          view_bundling="true"/>
        <FRAG2 frag_size="60000" down_thread="false" up_thread="false"/>
        <pbcast.STATE_TRANSFER down_thread="false" up_thread="false"
          use_flush="true" flush_timeout="3000"/>
        <pbcast.FLUSH down_thread="false" up_thread="false" timeout="20000"
          auto_flush_conf="false"/>
      </config>
    </attribute>

  </mbean>
```

5.4.1. Post Office Attributes

The following attributes are available to configure the Messaging Post Office:

DataSource

Specifies the data source used to persist post office mapping data.

SQLProperties

Specifies the DDL and DML for a particular database. If this attribute is not overridden, the default Hypersonic configuration will be used.

IMPORTANT

Hypersonic is not supported for production environments. This value should be changed for production use.

CreateTablesOnStartup

Specifies whether tables and index are created when the post office service is started. When set to `true`, the post office will create tables and indexes on startup. If tables or indexes already exist, a

SQLException is thrown by the JDBC driver, but this is ignored by the Persistence Manager and the operation continues unhindered. The default value is **true**.

DetectDuplicates

Specifies whether the post office will detect duplicate messages sent when delivery is restricted on a new node after server failure. When set to **true**, the post office detects duplicate messages . The default value is **true**.

IDCacheSize

Specifies the number of messages IDs the ID Cache should hold. If server failover occurs, the ID Cache is accessed as part of the process to prevent duplicate messages being sent after failover occurs. The default value is **500** (messages).

PostOfficeName

Specifies the name of the post office.

NodeIDView

Returns the node IDs of all nodes in a cluster as a set.

GroupName

Specifies the post office name to link with other identically named post offices.

NOTE

Post offices with the same group name will form a cluster together. Match post office group names to form a cluster with particular post offices.

Clustered

Specifies whether the post office will join a cluster to form distributed queues and topics. If set to **false**, all cluster-related attributes will be ignored.

StateTimeout

Specifies the maximum period of time the post office will wait to receive group state when a node joins a pre-existing cluster. The default value is **30000** milliseconds.

CastTimeout

Specifies the maximum time the post office will wait for a reply casting message synchronously. The default value is **30000** milliseconds.

FailoverOnNodeLeave

Specifies how connections are handled when a server node shuts down. If set to **true**, connections associated with the cleanly-terminated server node will failover onto another node. The default value is **false**.

MaxConcurrentReplications

Specifies the maximum number of concurrent replication reply requests to return before requests are blocked . This prevents JGroups overloading. The default value is **50**.

NOTE

You should not alter the default value for MaxConcurrentReplications. The default value is the optimal setting.

ControlChannelConfig

Specifies JGroups stack configuration for the control channel. The control channel sends requests to and receives responses from other nodes in the cluster.

NOTE

JBoss Messaging uses JGroups for all group management. Standard JGroups configuration is used.

DataChannelConfig

Specifies JGroups stack configuration for the data channel. The data channel sends and receives messages to and from other nodes in the cluster, and replicates session data.

NOTE

JBoss Messaging uses JGroups for all group management. Standard JGroups configuration is used.

5.5. CONFIGURING THE PERSISTENCE MANAGER

JBoss Messaging ships with a JDBC Persistence Manager, which handles message data persistence in a relational database accessed via JDBC. The Persistence Manager can be plugged into the Messaging server, which allows additional implementations to persist message data in non-relational stores, and file stores.

Persistent service configuration details are grouped in **`<database type>-persistence-service.xml`**. JBoss Messaging ships with the **`hsqldb-persistence-service.xml`** file by default, which configures the Messaging server to use the Hypersonic database instance included by default with any JBoss Enterprise Application Server instance.

WARNING

Hypersonic is not supported for use in a production environment.

JBoss Messaging also ships with Persistence Manager configurations for MySQL, Oracle, PostgreSQL, Sybase, Microsoft SQL Server, and DB2. The example configuration files (such as **`mysql-persistence-service.xml`** and **`ndb-persistence-service.xml`**) are available from the **`jboss-as/docs/examples/jms`** directory of the release bundle.

The JDBC Persistence Manager uses standard SQL as its Data Manipulation Language (DML), so writing a Persistence Manager configuration for another database type is a matter of changing the configuration's Data Definition Language (DDL), which usually differs on a per-database basis.

JBoss Messaging also ships with a Null Persistence Manager configuration option, which can be used when persistence is not required.

The following code is the default Hypersonic persistence manager configuration:

```
<mbean code="org.jboss.messaging.core.jmx.JDBCPersistenceManagerService"
  name="jboss.messaging:service=PersistenceManager"
  xmbean-dd="xmdesc/JDBCPersistenceManager-xmbean.xml">

  <depends>jboss.jca:service=DataSourceBinding,name=DefaultDS</depends>

  <depends optional-attribute-name="TransactionManager">
    jboss:service=TransactionManager
  </depends>

  <!-- The datasource to use for the persistence manager -->

  <attribute name="DataSource">java:/DefaultDS</attribute>

  <!-- If true will attempt to create tables and indexes on every start-up
-->

  <attribute name="CreateTablesOnStartup">true</attribute>

  <!-- If true then will use JDBC batch updates -->

  <attribute name="UsingBatchUpdates">false</attribute>

  <!-- The maximum number of parameters to include in a prepared statement
-->

  <attribute name="MaxParams">500</attribute>
</mbean>
```

IMPORTANT

The maximum size of Sybase database text and image data types is set to **2 kilobytes** by default. Any message that exceeds this limit is truncated, without any information or warning. Set the **@@TEXTSIZE** database parameter to a higher value to prevent potential truncation.

Truncation may also occur in the Microsoft SQL Server if **@@TEXTSIZE** value is set to a lesser value than the default value. For further information, see http://jira.jboss.com/jira/browse/SOA-554.

IMPORTANT

Microsoft SQL Server does not automatically unallocate hard drive space when data is deleted from a database. When the hard drive database space is used as a data store for a service that temporarily stores many records (such as a messaging service), the disk space will quickly become much greater than the amount of data actually being stored.

Database administrators must implement database maintenance plans to ensure that the unused space is reclaimed. Refer to your Microsoft SQL Server documentation for the DBCC commands **`ShrinkDatabase`** and **`UpdateUsage`** for guidance reclaiming the unused space. For further information about this issue, see https://jira.jboss.org/jira/browse/SOA-629

5.5.1. PersistenceManager MBean Attributes

The following attributes are available to configure the Persistence Manager MBean:

CreateTablesOnStartup

Specifies whether tables and index creation is attempted when the Persistence Manager is started. When set to **`true`** (default), the persistence manager will attempt to create tables (and indexes) on startup. If tables or indexes already exist, a **`SQLException`** will be thrown by the JDBC driver and ignored by the persistence manager, allowing it to continue unhindered.

UsingBatchUpdates

Specifies whether multiple database updates are grouped in batches to improve performance. Set this value to **`true`** if your database supports JDBC batch updates.. The default value is **`false`**.

UsingBinaryStream

Specifies whether messages are stored and read with a JDBC binary stream, instead of via **`getBytes()`** and **`setBytes()`**. Set this value to **`false`** if your database must use **`getBytes()`** and **`setBytes()`**. The default value is **`true`**.

UsingTrailingByte

Specifies how Sybase database BLOBs containing trailing zeroes are handled. When set to **`true`**, a trailing non-zero byte is added to each BLOB before persistence, and removed from the BLOB following persistence, preventing truncation by the database. The default value is **`false`**

NOTE

Certain versions of Sybase truncate a BLOB with trailing zeros. This attribute is only required if you are running a Sybase database.

SupportsBlobOnSelect

Specifies how BLOBs are inserted into certain database types. When set to **`false`**, two-stage insertion will be used. The default value is **`true`**.

NOTE

Certain databases, specifically Oracle, do not allow BLOB insertion via an `INSERT INTO ... SELECT FROM` statement, and require two-stage conditional message insertion. Set this attribute to `false` if you are running an Oracle database, or other database with this requirement.

SQLProperties

Specifies the DDL and DML for a particular database. If a particular DDL or DML statement is not overridden, the default Hypersonic configuration will be used for that statement.

MaxParams

Specifies the maximum number of parameters allowed per prepared statement while loading messages. The default value is `500`.

UseNDBFailoverStrategy

Specifies whether a SQL statement is re-executed in the event a database transation commit fails in a clustered environment. If set to `true`, the SQL statement is re-executed in the event that the commit fails. If a further error occurs, the persistence manager assumes the error is due to the previous transaction having committed successfully, and ignores the error. By default, this attribute is set to `false`.

NOTE

When some databases, such as MySQL, run in clustered environments, they can fail during database transaction commits. If this occurs, the final transaction state is unknown.

5.6. CONFIGURING THE JMS USER MANAGER

The JMS User Manager maps preconfigured client IDs to users. It also manages user and role tables, depending on the configured login module.

The following is an example `JMSUserManager` configuration:

```
<mbean code="org.jboss.jms.server.plugin.JDBCJMSUserManagerService"
  name="jboss.messaging:service=JMSUserManager"
  xmbean-dd="xmdesc/JMSUserManager-xmbean.xml">
  <depends>jboss.jca:service=DataSourceBinding,name=DefaultDS</depends>
  <depends optional-attribute-name="TransactionManager">
    jboss:service=TransactionManager
  </depends>
  <attribute name="DataSource">java:/DefaultDS</attribute>
  <attribute name="CreateTablesOnStartup">true</attribute>
  <attribute name="SqlProperties">
    CREATE_USER_TABLE=CREATE TABLE JBM_USER (USER_ID VARCHAR(32) NOT NULL,
      PASSWD VARCHAR(32) NOT NULL, CLIENTID VARCHAR(128),
      PRIMARY KEY(USER_ID)) ENGINE = INNODB

    CREATE_ROLE_TABLE=CREATE TABLE JBM_ROLE (ROLE_ID VARCHAR(32) NOT NULL,
      USER_ID VARCHAR(32) NOT NULL, PRIMARY KEY(USER_ID, ROLE_ID))
      ENGINE = INNODB

    SELECT_PRECONF_CLIENTID=SELECT CLIENTID FROM JBM_USER WHERE USER_ID=?

    POPULATE.TABLES.1=INSERT INTO JBM_USER (USER_ID,PASSWD,CLIENTID)
      VALUES ('jdoe','jdoepw','jdoe-id')
  </attribute>
</mbean>
```

5.6.1. JMSUserManager Managed Bean Attributes

CreateTablesOnStartup

Specifies whether tables and index creation is attempted when the JMSUserManager MBean is started. When set to **`true`** (default), the JMSUserManager will attempt to create tables (and indexes) on startup. If tables or indexes already exist, a **`SQLException`** will be thrown by the JDBC driver and ignored by the persistence manager, allowing it to continue unhindered.

UsingBatchUpdates

Specifies whether multiple database updates are grouped in batches to improve performance. Set this value to **`true`** if your database supports JDBC batch updates. The default value is **`false`**.

SQLProperties

Specifies the DDL and DML for a particular database. If a particular DDL or DML statement is not overridden, the default Hypersonic configuration will be used for that statement.

You can also insert default user and role property data by prefixing the data with **`POPULATE.TABLES`**.

5.7. CONFIGURING DESTINATIONS

5.7.1. Pre-configured destinations

JBoss Messaging ships with a default set of preconfigured destinations that are deployed at server start-up. The configuration information for these destinations can be found in the following section of **`destinations-service.xml`**:

```
<!--
      The Default Dead Letter Queue. This destination is a dependency of
an EJB MDB container.
   -->

   <mbean code="org.jboss.jms.server.destination.QueueService"
      name="jboss.messaging.destination:service=Queue,name=DLQ"
      xmbean-dd="xmdesc/Queue-xmbean.xml">
      <depends optional-attribute-
name="ServerPeer">jboss.messaging:service=ServerPeer</depends>
      <depends>jboss.messaging:service=PostOffice</depends>
   </mbean>

   <!--
```

```
      The Default Expiry Queue.
   -->

   <mbean code="org.jboss.jms.server.destination.QueueService"
      name="jboss.messaging.destination:service=Queue,name=ExpiryQueue"
      xmbean-dd="xmdesc/Queue-xmbean.xml">
      <depends optional-attribute-
name="ServerPeer">jboss.messaging:service=ServerPeer</depends>
      <depends>jboss.messaging:service=PostOffice</depends>
   </mbean>
```

5.7.2. Configuring Queues

5.7.2.1. Queue MBean Attributes

Name

Defines the queue name.

JNDIName

Defines the JNDI name that binds the queue.

DLQ

Defines the DLQ (Dead Letter Queue) for this queue and overrides any value set in the Server Peer configuration file.

ExpiryQueue

Defines the expiry queue and overrides any value set in the Server Peer configuration file.

RedeliveryDelay

Defines the redelivery delay to be applied to this queue and overrides any value set in the Server Peer configuration file.

MaxDeliveryAttempts

Defines the maximum number of times message delivery is attempted before the message is sent to the DLQ, if configured. The default value, `-1`, means that the value from the Server Peer configuration file is used. Any other setting will override the value set in the Server Peer configuration file.

CreatedProgrammatically

Returns `true` if the queue was created programmatically.

MessageCount

Returns the total number of messages in the queue. That is, the number of messages being scheduled plus the number being delivered, plus the number not being delivered.

ScheduledMessageCount

Returns the number of *scheduled messages* in the queue. This is the number of messages scheduled to be delivered at a later date.

Scheduled delivery lets you specify the earliest time at which a particular message will be delivered.

For example, you can send a message now, and specify that it will not be delivered for two hours. To do so, set the following in the message header:

```
long now = System.currentTimeMillis();

Message msg = sess.createMessage();

msg.setLongProperty(JBossMessage.JMS_JBOSS_SCHEDULED_DELIVERY_PROP_NAME,
  now + 1000 * 60 * 60 * 2);

prod.send(msg);
```

MaxSize

Specifies the maximum number of messages that can be held in a queue. Any excess messages will be dropped. The default value is **`-1`**, which is unbounded.

Clustered

This attribute must be set to **`true`** if the destination is clustered.

MessageCounter

Each queue maintains a message counter.

MessageCounterStatistics

The statistics for the message counter.

MessageCounterHistoryDayLimit

The maximum number of days for which to hold message counter history. Overrides any value set in the Server Peer configuration file.

ConsumerCount

The number of consumers currently consuming from the queue.

DropOldMessageOnRedeploy

Specifies how queue services with clustered attributes that differ from previously deployed attributes are handled. If set to **`true`**, all remaining messages in the queue are deleted after the queue service re-deployment if the queue service attribute contains a different clustered attribute. If set to **`false`** (default), all messages are reserved.

WARNING

When you re-deploy a destination, you must shut down all the nodes in the cluster, make proper configuration changes, and then restart the nodes.

Redeploying from a non-clustered to a clustered queue requires you set the clustered attribute to `true`, and add the queue service configuration to each node.

Redeploying from a clustered to a non-clustered queue requires you set the clustered attribute to `false` in one of the queue configurations and delete all other queues in the cluster.

5.7.2.1.1. Destination Security Configuration

<SecurityConfig> determines which roles can read, write and create upon the destination. It uses the same syntax and semantics as JBossMQ destination security configuration.

```
<SecurityConfig>
  <security>
    <role read="true" write="true" create="true"/>
  </security>
<SecurityConfig>
```

The <SecurityConfig> element must contain one <security> element, which can contain multiple <role> elements. A <role> element defines the access type for that particular role using the following attributes:

read

Specifies the role can create consumers, receive messages, and browse the destination.

write

Specifies the role can create producers, or send messages to the destination.

create

Specifies the role can create durable subscriptions on this destination.

NOTE

Configuring security for a destination is optional. If a `SecurityConfig` element is not specified, then the default security configuration from the Server Peer will be used instead.

5.7.2.1.2. Destination paging parameters

Pageable Channels is a JBoss Messaging feature that lets you specify a maximum number of messages to be stored in memory at one time, on a queue-by-queue or topic-by-topic basis. JBoss Messaging then pages messages to and from storage transparently in blocks. This allows queues and subscriptions to grow to very large sizes without any degradation in performance as channel size increases.

The individual parameters associated with pageable channels are as follows:

FullSize

Specifies the maximum number of messages held by the queue or topic subscription in memory at any one time. The actual queue can hold more messages, but these are paged to and from storage as messages are added or consumed. If no value is specified, the default is **`75000`**.

PageSize

Specifies the maximum number of messages that are pre-loaded per operation when loading messages from the queue or subscription. If no value is specified, the default is **`2000`**.

DownCacheSize

Specifies the maximum number of messages the Down Cache holds before the messages are flushed to storage. The default value is **`2000`** messages.

When messages are paged to storage from the queue, they enter a *Down Cache* before being written to storage. This enables the write to occur as a single operation, which aids performance.

NOTE

Paging parameters for temporary queues must be specified on the appropriate connection factory. Refer to Section 5.8, “Configuring Connection Factories” for detailed information about the different connection factories available.

5.7.2.1.3. Queue Managed Bean Operations

RemoveAllMessages

Removes (and deletes) all messages from the queue.

IMPORTANT

This will permanently delete all messages from the queue; use this operation with caution.

ListAllMessages

Lists all messages currently in the queue. Using a JMS selector as an argument in this operation lets you retrieve a subset of the messages in the queue that match the given criteria.

ListDurableMessages

Lists all *durable* messages in the queue. Using a JMS selector as an argument in this operation lets you retrieve a subset of messages in the queue that match the given criteria.

ListNonDurableMessages

Lists all *non-durable* messages in a queue. Using a JMS selector as an argument in this operation lets you retrieve a subset of messages in the queue that match the given criteria.

ResetMessageCounter

Resets the message counter to zero.

ResetMessageCounterHistory

Resets the message counter history.

ListMessageCounterAsHTML

Lists the message counter in HTML format.

ListMessageCounterHistoryAsHTML

Lists the message counter history in HTML format.

5.7.3. Configuring Topics

5.7.3.1. Topic Managed Bean Attributes

Name

Defines the name of the topic.

JNDIName

Defines the JNDI location where the topic is bound.

DLQ

Defines the Dead Letter Queue (DLQ) used for this topic and overrides any value set in the Server Peer configuration file.

ExpiryQueue

Defines the expiry queue used for this topic and overrides any value set in the Server Peer configuration file.

RedeliveryDelay

Defines the delay period between redelivery attempts for this topic and overrides any value set in the Server Peer configuration file.

MaxDeliveryAttempts

Defines the maximum number of times message delivery will be attempted before the message is sent to the DLQ, if configured. The default value is `-1`, which specifies that the value from the Server Peer configuration file be used. Any other setting overrides the Server Peer value.

CreatedProgrammatically

Returns `true` if the topic was created programmatically.

MaxSize

Specifies the maximum number of messages that can be held in a topic subscription. Any excess messages will be dropped from the topic. The default value is `-1`, which applies no size restriction.

Clustered

Set this to `true` if your destination is clustered.

MessageCounterHistoryDayLimit

Defines the maximum number of days to retain message counter history, and overrides any value set in the Server Peer configuration file.

MessageCounters

Returns a list of message counters for the topic's subscriptions.

AllMessageCount

Returns the total number of messages in all subscriptions belonging to the topic.

DurableMessageCount

Returns the total number of *durable* messages in all subscriptions belonging to this topic.

NonDurableMessageCount

Returns the total number of *non-durable* messages in all subscriptions belonging to this topic.

DropOldMessageOnRedeploy

Specifies how queue services with clustered attributes that differ from previously deployed attributes are handled. If set to `true`, all remaining messages in the queue are deleted after the queue service re-deployment if the queue service attribute contains a different clustered attribute. If set to `false` (default), all messages are reserved.

WARNING

When you re-deploy a destination, you must shut down all the nodes in the cluster, make proper configuration changes, and then restart the nodes.

Redeploying from a non-clustered to a clustered queue requires you set the clustered attribute to `true`, and add the queue service configuration to each node.

Redeploying from a clustered to a non-clustered queue requires you set the clustered attribute to `false` in one of the queue configurations and delete all other queues in the cluster.

AllSubscriptionsCount

Returns a count of all subscriptions belonging to this topic.

DurableSubscriptionsCount

Returns a count of all durable subscriptions belonging to this topic.

NonDurableSubscriptionsCount

Returns a count of all non-durable subscriptions belonging to this topic.

5.7.3.1.1. Destination Security Configuration

<SecurityConfig> determines which roles can read, write and create upon the destination. It uses the same syntax and semantics as JBossMQ destination security configuration.

The <SecurityConfig> element must contain one <security> element, which can contain multiple <role> elements. A <role> element defines the access type for that particular role using the following attributes:

read

Specifies the role can create consumers, receive messages, and browse the destination.

write

Specifies the role can create producers, or send messages to the destination.

create

Specifies the role can create durable subscriptions on this destination.

NOTE

Configuring security for a destination is optional. If a `SecurityConfig` element is not specified, then the default security configuration from the Server Peer will be used instead.

5.7.3.1.2. Destination paging parameters

Previously, for an application to support a queue or subscription, the queue needed to be stored entirely in memory. This was not always possible for very large queues or subscriptions.

Pageable Channels is a new JBoss Messaging feature that lets you specify a maximum number of messages to be stored in memory at one time, on a queue-by-queue or topic-by-topic basis. JBoss Messaging then pages messages to and from storage transparently in blocks. This allows queues and subscriptions to grow to very large sizes without any degradation in performance as channel size increases. It has been tested with queues in excess of ten million 2 kilobyte messages on very basic hardware, and has the potential to scale to much greater message numbers.

The individual parameters associated with pageable channels are as follows:

FullSize

Specifies the maximum number of messages held by the queue or topic subscription in memory at any one time. The actual queue can hold more messages, but these are paged to and from storage as messages are added or consumed. If no value is specified, the default is `75000`.

PageSize

Specifies the maximum number of messages that are pre-loaded per operation when loading messages from the queue or subscription. If no value is specified, the default is `2000`.

DownCacheSize

Specifies the maximum number of messages the Down Cache holds before the messages are flushed to storage. The default value is `2000` messages.

When messages are paged to storage from the queue, they enter a *Down Cache* before being written to storage. This enables the write to occur as a single operation, which aids performance.

NOTE

Paging parameters for temporary queues must be specified on the appropriate connection factory. See the section on Connection Factory Configuration for details.

5.7.3.2. Topic Managed Bean Operations

RemoveAllMessages

Removes (and deletes) all messages from subscriptions that belong to this topic.

IMPORTANT

This will permanently delete all messages from the topic; use this operation with caution.

ListAllMessages

Lists all messages belonging to a specified subscription. Using a JMS selector as an argument in this operation lets you retrieve a subset of messages in the queue that match the given criteria.

ListDurableMessages

Lists all durable messages belonging to the specified subscription. Using a JMS selector as an argument in this operation lets you retrieve a subset of messages in the queue that match the given criteria.

ResetMessageCounter

Resets the message counter to zero.

ResetMessageCounterHistory

Resets the message counter history.

ListAllSubscriptionsAsHTML

Lists all subscriptions belonging to this topic in HTML format.

ListDurableSubscriptionsAsHTML

Lists all durable subscriptions belonging to this topic in HTML format.

ListNonDurableSubscriptions

Lists all non-durable messages belonging to the specified subscription. Using a JMS selector as an argument in this operation lets you retrieve a subset of messages in the queue that match the given criteria.

ListNonDurableSubscriptionsAsHTML

Lists all non-durable subscriptions belonging to this topic in HTML format.

5.8. CONFIGURING CONNECTION FACTORIES

JBoss Messaging is configured by default to bind two connection factories in JNDI upon startup.

The first connection factory is the default, non-clustered connection factory. This connection factory is provided to maintain compatibility with applications originally written against JBossMQ, which does not include automatic failover or load balancing. If you do not require client-side automatic failover or load balancing, then you should use this first connection factory.

The first connection factory is bound into the following JNDI contexts:

- **`/ConnectionFactory`**
- **`/XAConnectionFactory`**
- **`java:/ConnectionFactory`**
- **`java:/XAConnectionFactory`**.

The second connection factory is the default clustered connection factory, which is bound into the following JNDI contexts:

- **`/ClusteredConnectionFactory`**
- **`/ClusteredXAConnectionFactory`**
- **`java:/ClusteredConnectionFactory`**
- **`java:/ClusteredXAConnectionFactory`**

If you want to provide a default client ID for a connection factory, or bind a connection factory to a different JNDI locationConsider, then configure and deploy additional connection factories. To deploy a new connection factory, configure a new **`ConnectionFactory`** managed bean in **`connection-factories-service.xml`**.

You can also create a new service deployment descriptor, **`<`*`name`*`>-service.xml`**, and deploy it in **`$JBOSS_HOME/server/messaging/deploy`**.

Enable support for automatic failover or load balancing by setting the relevant attributes in your connection factory:

Example 5.1. Connection Factory

This example connection factory creates a connection factory with the preconfigured client ID **`myClientID`**, which is bound to two locations in the JNDI tree: **`/MyConnectionFactory`** and **`/factories/cf`**.

The example overrides the following default values:

- **`PreFetchSize`**
- **`DefaultTempQueueFullSize`**
- **`DefaultTempQueuePageSize`**
- **`DefaultTempQueueDownCacheSize`**

- **`DupsOKBatchSize`**
- **`SupportsFailover`**
- **`SupportsLoadBalancing`**
- **`LoadBalancingFactory`**

The connection factory uses the default remoting connector. To use a different remoting connector with the connection factory, change the **`Connector`** attribute to specify the service name of the connector you wish to use.

```
<mbean code="org.jboss.jms.server.connectionfactory.ConnectionFactory"
  name="jboss.messaging.connectionfactory:service=MyConnectionFactory"
  xmbean-dd="xmdesc/ConnectionFactory-xmbean.xml">
  <depends optional-attribute-name="ServerPeer">
    jboss.messaging:service=ServerPeer
  </depends>
  <depends optional-attribute-name="Connector">
    jboss.messaging:service=Connector,transport=bisocket
  </depends>
  <depends>jboss.messaging:service=PostOffice</depends>

  <attribute name="JNDIBindings">
    <bindings>
    <binding>/MyConnectionFactory</binding>
    <binding>/factories/cf</binding>
    </bindings>
  </attribute>

  <attribute name="ClientID">myClientID</attribute>

  <attribute name="SupportsFailover">true</attribute>

  <attribute name="SupportsLoadBalancing">false</attribute>

  <attribute name="LoadBalancingFactory">
    org.acme.MyLoadBalancingFactory
  </attribute>

  <attribute name="PrefetchSize">1000</attribute>

  <attribute name="SlowConsumers">false</attribute>

  <attribute name="StrictTck">true</attribute>

  <attribute name="SendAcksAsync">false</attribute>

  <attribute name="DefaultTempQueueFullSize">50000</attribute>

  <attribute name="DefaultTempQueuePageSize">1000</attribute>

  <attribute name="DefaultTempQueueDownCacheSize">1000</attribute>

  <attribute name="DupsOKBatchSize">10000</attribute>
</mbean>
```

5.8.1. ConnectionFactory Managed Bean Attributes

ClientID

You can preconfigure a connection factory with a client ID. Any connection created via this connection factory will obtain this client ID.

JNDIBindings

Lists available JNDI bindings for this connection factory.

PrefetchSize

Specifies how many messages the window holds at once, for consumer flow control. The window size determines the number of messages a server can send to a consumer without blocking. Each consumer maintains a buffer of messages from which it consumes.

Transmission Control Protocol (TCP) implements its own additional flow control. Message consumption can also be blocked if the TCP window size is smaller than the ***`PrefetchSize`*** parameter.

SlowConsumers

Specifies whether the allowable buffer size for slow consumers is reduced. Reducing the buffer size for slow consumers results in minimized to increase the potential for messages to be consumed by faster consumers. It is not possible to totally disable buffering, however, setting the **`SlowConsumers`** attribute to **`true`** will reduce the buffer size. Setting this attribute to **`true`** is equivalent to setting **`PrefetchSize`** to **`1`** which is the lowest possible value available.

StrictTck

Enables strict JMS behavior if the attribute is set to **`true`**. Strict JMS behavior is required by the Technology Compatibility Kit (TCK).

SendAcksAsync

Specifies acknowledgments are sent asynchronously if the attribute is set to **`true`**. This can improve performance, particularly if **`auto_acknowledge`** mode is active.

DefaultTempQueueFullSize

Optional attribute that specifies the paging parameters for temporary full size queue destinations, which are scoped to connections created with this connection factory. The default value is **`200000`**. For more information about these attributes, refer to Section 5.7.3.1.2, “Destination paging parameters”.

DefaultTempQueuePageSize

Optional attribute that specifies the paging parameters for temporary page size destinations, which are scoped to connections created with this connection factory. The default value is **`2000`**. For more information about these attributes, refer to Section 5.7.3.1.2, “Destination paging parameters”.

DefaultTempQueueDownCacheSize

Optional attribute that specifies the paging parameters for temporary down cache size destinations, which are scoped to connections created with this connection factory. The default value is **`2000`**. For more information about these attributes, refer to Section 5.7.3.1.2, “Destination paging parameters”.

DupsOKBatchSize

Specifies the number of **`DUPS_OK_ACKNOWLEDGE`** acknowledgments that are buffered locally before they are sent. The default value is **`2000`**.

SupportsLoadBalancing

Specifies whether client-side load balancing is enabled for the connection factory on clustered installations. If load balancing is enabled, any connection created by that connection factory will be load-balanced across the nodes of a cluster. A connection created on a particular node remains on that node. The default value is **`false`**.

SupportsFailover

Specifies whether client-side automatic failover is enabled for the connection factory on clustered installations. If automatic failover is enabled, JBoss Messaging will automatically and transparently failover to another node in the cluster when a connection problem is detected. The default value is **`false`**.

NOTE

When automatic failover is disabled, the user code is responsible for catching connection exceptions in synchronous JMS operations, and a JMS **`ExceptionListener`** must be installed to catch exceptions asynchronously. When an exception is caught, the client-side code must lookup a new connection factory via HAJNDI and recreate the connection.

DisableRemotingChecks

Specifies whether the connection factory checks that the corresponding JBoss Remoting Connector uses sensible values. JBoss Messaging is very sensitive to these values, and there is rarely any need to change them. To disable this sanity checking, set **`DisableRemotingChecks`** to **`false`**. The default value is **`true`**.

WARNING

Do not disable the remoting checks; system instability.

LoadBalancingFactory

Specifies the client-side load balancing factory implementation used by the connection factory. The value must correspond to the name of a class that implements the interface **`org.jboss.jms.client.plugin.LoadBalancingFactory`**.

The default value is **`org.jboss.jms.client.plugin.RoundRobinLoadBalancingFactory`**, which load-balances connections across the cluster in a round-robin fashion.

Connector

Specifies the remoting connector used by the connection factory. Different connection factories can use different connectors, so you can deploy one connection factory that uses the HTTP transport to communicate with the server, and another that uses the bisocket transport to communicate.

EnableOrderingGroup

Specifies whether strict message ordering is enabled on the `ConnectionFactory`. If set to `true`, any messages sent from producers which are created from the enabled connection factory become ordering group messages. The default value for this parameter is `false`.

DefaultOrderingGroupName

Specifies the default name for the message ordering group. The specified name will take effect once the `EnableOrderingGroup` parameter is set to `true`. If this attribute is missing, the group name will be generated automatically.

5.9. CONFIGURING THE REMOTING CONNECTOR

JBoss Messaging uses JBoss Remoting for all communication between the client and the server.

For further information about JBoss Remoting configuration and capabilities, see the [Remoting] chapter in the *JBoss Administration and Configuration Guide*

The default configuration includes one remoting connector, which is used by the single default connection factory. Each connection factory can be configured to use a different connector.

The default connector is configured to use the remoting bisocket transport, a TCP socket-based transport that listens and accepts connections only on the server side. That is, connections are always initiated from the client side. This connector is ideal for typical firewall scenarios, where only inbound connections are allowed on the server, or where only outbound connections are allowed from the client.

The bisocket transport can be configured to use SSL when a higher level of security is required.

The other supported transport is the HTTP transport, which uses the Hypertext Transfer Protocol to communicate between client and server. The client periodically polls the server for messages to receive data. This transport is ideal when a firewall between server and client allows only incoming HTTP traffic on the server. Because of its polling behavior and the limitations of HTTP, this transport does not perform as well as the bisocket transport. It is not designed to handle high-load situations.

No other remoting transports are currently supported by JBoss Messaging.

Remoting configuration details can be seen in **`$JBOSS_HOME/server/$SERVER/deploy/messaging/remoting-bisocket-service.xml`**. The following code is an example of a bisocket remoting configuration:

Example 5.2. Bisocket Remoting Configuration

```
<?xml version="1.0" encoding="UTF-8"?>

<!--
     Standard bisocket-based Remoting service deployment descriptor.

     $Id: remoting-bisocket-service.xml 3981 2008-03-28 18:00:41Z timfox
```

```
$
 -->

<server>

   <!-- Standard bisocket connector - the bisocket transport only opens
connection from client->server
        so can be used with firewalls where only outgoing connections
are allowed.
        For examples of HTTP and SSL transports see docs/examples -->
   <mbean code="org.jboss.remoting.transport.Connector"
          name="jboss.messaging:service=Connector,transport=bisocket"
          display-name="Bisocket Transport Connector">
      <attribute name="Configuration">
         <config>
            <invoker transport="bisocket">

               <!-- There should be no reason to change these
parameters - warning!
                    Changing them may stop JBoss Messaging working
correctly -->
               <attribute name="marshaller"
isParam="true">org.jboss.jms.wireformat.JMSWireFormat</attribute>
               <attribute name="unmarshaller"
isParam="true">org.jboss.jms.wireformat.JMSWireFormat</attribute>
               <attribute name="dataType"
isParam="true">jms</attribute>
               <attribute name="socket.check_connection"
isParam="true">false</attribute>
               <attribute
name="serverBindAddress">${jboss.bind.address}</attribute>
               <attribute
name="serverBindPort">${jboss.messaging.connector.bisocket.port:4457}
</attribute>
               <attribute name="clientSocketClass"
isParam="true">org.jboss.jms.client.remoting.ClientSocketWrapper</attrib
ute>
               <attribute
name="serverSocketClass">org.jboss.jms.server.remoting.ServerSocketWrapp
er</attribute>
               <attribute
name="onewayThreadPool">org.jboss.jms.server.remoting.DirectThreadPool</
attribute>

               <!-- the following parameters are useful when there is a
firewall between client and server. Uncomment them if so.-->
               <!--
               <attribute name="numberOfCallRetries"
isParam="true">1</attribute>
               <attribute name="pingFrequency"
isParam="true">214748364</attribute>
               <attribute name="pingWindowFactor"
isParam="true">10</attribute>
               <attribute name="generalizeSocketException"
isParam="true">true</attribute>
               -->
               <!-- Now remoting supports socket write timeout
configuration. Uncomment this if you need it. -->
               <!--
               <attribute name="writeTimeout"
isParam="true">30000</attribute>
               -->

               <!-- End immutable parameters -->

               <attribute name="stopLeaseOnFailure"
isParam="true">true</attribute>

               <!-- Periodicity of client pings. Server window by
default is twice this figure -->
               <attribute name="clientLeasePeriod"
isParam="true">10000</attribute>
               <attribute name="validatorPingPeriod"
isParam="true">10000</attribute>
               <attribute name="validatorPingTimeout"
isParam="true">5000</attribute>

               <attribute name="failureDisconnectTimeout"
isParam="true">0</attribute>
               <attribute name="callbackErrorsAllowed">1</attribute>
               <attribute
name="registerCallbackListener">false</attribute>
               <attribute name="useClientConnectionIdentity"
isParam="true">true</attribute>

             <attribute name="timeout" isParam="true">0</attribute>

               <!-- Max Number of connections in client pool. This
should be significantly higher than
                    the max number of sessions/consumers you expect -->
               <attribute name="JBM_clientMaxPoolSize"
isParam="true">200</attribute>

               <!-- The maximum time to wait before timing out on
trying to write a message to socket for delivery -->
               <attribute name="callbackTimeout">10000</attribute>

               <!-- Use these parameters to specify values for binding
and connecting control connections to
                    work with your firewall/NAT configuration
               <attribute name="secondaryBindPort">xyz</attribute>
               <attribute name="secondaryConnectPort">abc</attribute>
               -->

            </invoker>
            <handlers>
               <handler
subsystem="JMS">org.jboss.jms.server.remoting.JMSServerInvocationHandler
</handler>
            </handlers>
         </config>
```

```
        </attribute>
    </mbean>

</server>
```

There are restricted attributes that should not be changed unless you are absolutely confident you understand the impact of the changes. The following attributes are safe to change and configure to the requirements of your project:

clientLeasePeriod

Clients periodically return *heartbeats* to the server to confirm that they are still active. If the server does not receive a heartbeat after a certain period of time, it will close down the connection and remove all resources that correspond to the client's session. The **`clientLeasePeriod`** determines the period of time between heartbeats, in milliseconds. The default value is **`10000`**.

By default, the server closes a client if it does not receive a heartbeat within double the **`clientLeasePeriod`**. In reality, the period is automatically resized according to system load.

numberOfRetries

The number of seconds JBoss Remoting blocks on the client pool while waiting for a connection to become available. If you have a very large number of sessions concurrently accessing the server from a client and cannot obtain connections from the pool, you may want to increase this value.

clientMaxPoolSize

JBoss Remoting maintains a client-side pool of TCP connections on which to service requests. If you have a large number of sessions concurrently accessing the server from a client and cannot obtain connections from the pool, you may want to increase this value.

secondaryBindPort

The bisocket transport uses control connections to pass control messages between server and client. This attribute defines the address to which the secondary **`ServerSocket`** is bound. To work behind a firewall, you may need to specify a particular value for your firewall configuration.

secondaryConnectPort

The port that the client uses to connect. Specify this to let your client work with NAT routers.

maxPoolSize

The number of threads used on the server side to service requests.

By default, JBoss Messaging binds to **`${jboss.bind.address}`**, which can be defined by running the **`./run.sh -c`** ***`[yourconfig]`*** **`-b`** ***`[yourIP]`*** command.

If necessary, you can change **`remoting-bisocket-service.xml`** to use a different communication port.

WARNING

Do not change values in the connector configuration other than those listed above. Changing other values can cause JBoss Messaging to stop functioning correctly.

5.10. SERVICEBINDINGMANAGER

The **`SeviceBindingManager`** provides multiple application server instances running on the same IP using different port ranges, which is useful during development. There are other ways to do this, but the **`ServiceBindingManager`** removes much hassle.

CHAPTER 6. CLUSTERING NOTES

To help locate clustering-related information, a summary of each consideration is provided in this part of the guide with links to the related components of JBoss Messaging.

6.1. UNIQUE SERVER PEER ID

In most cases, JBoss Messaging works in a clustered environment with minimal configuration changes. One crucial change that must be made is that every node is assigned a unique server ID.

Every deployed node must have a unique ID, including nodes that form a LAN cluster and nodes linked by message bridges.

The ***`ServerPeerID`*** attribute is used to set this information. Refer to Section 5.2, “ServerPeer attributes” for further information.

6.2. CLUSTERED DESTINATIONS

JBoss Messaging clusters Java Message Service (JMS) queues and topics transparently across the cluster. Messages sent to a distributed queue or topic on one node are consumable on other nodes. To make a particular destination clustered, the **`clustered`** attribute is used to set this functionality. Refer to Section 5.4.1, “Post Office Attributes” for further information.

JBoss Messaging balances messages between nodes and caters for consumers of varying speeds so processing load can be efficiently distributed across the cluster.

To disable message redistribution between nodes while retaining other characteristics of clustered destinations, do not specify the **`ClusterPullConnectionFactoryName`** attribute on the Server Peer. Refer to Section 5.2, “ServerPeer attributes” for full details about this attribute.

6.3. CLUSTERED DURABLE SUBSCRIPTIONS

JBoss Messaging durable subscriptions can be clustered in a way that allows multiple subscribers on multiple nodes to consume from one durable subscription. A durable subscription is clustered automatically, providing its topic is clustered.

For more information about configuring clustered topics and queues, refer to the ***`Clustered`*** attribute in Section 5.4.1, “Post Office Attributes”

6.4. CLUSTERED TEMPORARY DESTINATIONS

JBoss Messaging supports clustering of temporary topics and queues. All temporary topics and queues will be clustered if the Post Office is clustered.

For more information about configuring clustered topics and queues, refer to the ***`Clustered`*** attribute in Section 5.4.1, “Post Office Attributes” .

6.5. NON-CLUSTERED SERVERS

Set the ***`PostOffice`*** clustered attribute to `false` if you do not want all nodes to participate in a cluster, or if you do not want the server to be clustered.

For more information about configuring non-clustered server, refer to the various attributes in Section 5.4.1, “Post Office Attributes” .

6.6. MESSAGE ORDERING IN THE CLUSTER

To ensure messages are consumed in the same order they were produced, set strict JMS ordering by setting the `DefaultPreserveOrdering` Server Peer attribute to `true`. While set to `true`, messages cannot be distributed as freely around the cluster. The default value is `false`.

6.7. IDEMPOTENT OPERATIONS

A message is guaranteed to be persisted when the message sent to a persistent destination returns with no exception.

An exception does not guarantee the message was *not* persisted, because failure may have occurred between the message being persisted and a response being returned to the caller.

Applications must therefore be coded so that operations are *idempotent* — that is, operations can be repeated without causing the system to become inconsistent.

You can implement this behaviour on the application level by checking for duplicate messages and discarding them if the original message has been sent successfully. This *duplicate checking* is a powerful technique that can remove the need for XA transactions.

JBoss Messaging is configured by default to perform duplicate checking in a clustered environment.

Persistence considerations are located in Section 5.2.1, “ServerPeer methods”, Section 5.3, “Changing the Database”, Section 5.5, “Configuring the Persistence Manager”, and Section 8.1, “Message Bridge Overview”.

6.8. CLUSTERED CONNECTION FACTORIES

When `supportsLoadBalancing` is set to `true` in the connection factory, consecutive attempts to create connections will round-robin between available servers. The first node is chosen randomly.

When `supportsFailover` is set to `true`, failover will occur transparently and automatically whenever any connection error is detected.

For more information about configuring connection factories, refer to Section 5.8.1, “ConnectionFactory Managed Bean Attributes”.

CHAPTER 7. JBOSS MESSAGING XA RECOVERY CONFIGURATION

This section describes how to configure JBoss Transactions to handle XA recovery for JBoss Messaging resources in JBoss Enterprise Application Platform.

The JBoss Transactions Recovery Manager can be configured to continually poll for and recover JBoss Messaging XA resources. This provides a high level of transaction durability.

To enable JBoss Transactions Recovery Manager, add a line to **`$JBOSS_HOME/server/$PROFILE/conf/jbossts-properties.xml`**. The following code snippet includes the line required:

```
<properties depends="arjuna" name="jta">
  <!--
  Support subtransactions in the JTA layer?
  Default is NO.
  -->
  <property name="com.arjuna.ats.jta.supportSubtransactions" value="NO"/>
  <property name="com.arjuna.ats.jta.jtaTMImplementation"

value="com.arjuna.ats.internal.jta.transaction.arjunacore.TransactionManag
erImple"/>
  <property name="com.arjuna.ats.jta.jtaUTImplementation"

value="com.arjuna.ats.internal.jta.transaction.arjunacore.UserTransactionI
mple"/>
  <!--
      *** Add this line to enable recovery for JMS resources using
DefaultJMSProvider ***
  -->
  <property
name="com.arjuna.ats.jta.recovery.XAResourceRecovery.JBMESSAGING1"

value="org.jboss.jms.server.recovery.MessagingXAResourceRecovery;java:/Def
aultJMSProvider"/>

</properties>
```

Here, the Recovery Manager attempts to recover JMS resources via the JMS Provider Loader, **`DefaultJMSProvider`**.

`DefaultJMSProvider` ships with JBoss Enterprise Application Platform. It is defined in **`$JBOSS_HOME/server/$PROFILE/conf/jms-ds.xml`** (or, in a clustered environment, **`hajndi-jms-ds.xml`**). To perform recovery with a different JMS provider loader (for example, one that corresponds with a remote JMS Provider), add another line to the properties file and specify your remote provider instead of **`DefaultJMSProvider`**. Your provider's name should be listed in its managed bean configuration file.

Each provider requires a unique name, for example, **`com.arjuna.ats.jta.recovery.XAResourceRecovery.JBMESSAGING1`**, **`com.arjuna.ats.jta.recovery.XAResourceRecovery.JBMESSAGING2`**, etc.

Recovery should work with any JMS provider that implements recoverable XAResources (that is, it properly implements **`XAResource.recover()`**).

For the Recovery Manager to recover from any node of the cluster, you must add a line in **`hajndi-jms-ds.xml`** for every node of the cluster.

CHAPTER 8. JBOSS MESSAGING MESSAGE BRIDGE CONFIGURATION

8.1. MESSAGE BRIDGE OVERVIEW

JBoss Messaging includes a fully functional message bridge.

The bridge consumes messages from a source queue or topic and sends them to a target queue or topic, typically on a different server. The source and target servers do not need to be in the same cluster, so bridging is a reliable method of sending messages from one cluster to another (across a WAN, for example) and where the connection may be unreliable.

A bridge is deployed as a managed bean within any JBoss Enterprise Application Platform instance. To deploy, add the managed bean descriptor into the `deploy` directory of an Enterprise Application Platform configuration that contains JBoss Messaging.

The example in `$JBOSS_HOME/docs/examples/jboss-messaging-examples/bridge/` demonstrates a simple bridge deployed in JBoss Enterprise Application Platform and moving messages from the source to the target destination.

The bridge can also be used to retrieve messages from other non-JBoss Messaging JMS servers as long as they are JMS 1.1 compliant.

The bridge has built-in failure recovery; if the source or target server connection is lost, the bridge will attempt to reconnect to the source or target until it comes back online, at which point normal operation will resume.

The bridge can be configured to consume messages matching a particular JMS selector.

It can be configured to consume from a queue or a topic. When the bridge consumes from a topic, it can be configured to consume with a non-durable or a durable subscription.

The bridge can be configured to handle messages with one of three *quality of service* (QoS) levels:

Bridge QoS Levels

`QOS_AT_MOST_ONCE`

This mode specifies that messages will arrive at the destination once at the most. Messages are consumed from the source and acknowledged before they are sent to the destination. Messages can be lost if failure occurs between the message leaving the source and arriving at the destination. Messages will therefore be delivered once at most.

This mode is available for both persistent and non-persistent messages.

`QOS_DUPLICATES_OK`

This mode specifies that messages are consumed from the source and acknowledged after they have been successfully sent to the destination. If failure occurs between a message arriving, and being acknowledged by the destination, that message is sent a second time when the system recovers.

This mode is available for both persistent and non-persistent messages.

`QOS_ONCE_AND_ONLY_ONCE`

This mode specifies that messages will arrive exactly once. When the message source and destination are on the same JBoss Messaging server instance, the message can be sent and received in the same local transaction.

If the source and destination are on different servers, you can implement message high durability by using a JTA transaction controlled by JBoss Transactions JTA implementation. If JTA is required, both connection factories must be **`XAConnectionFactory`** implementations.

This mode is only available for persistent messages.

This mode requires logging on both the transaction manager and the resource side to support recovery. If you require this level of QOS, you must enable XA Recovery with JBoss Transactions.

NOTE

You may be able to apply *once and only once*semantics to a specific application, without setting **`QOS_ONCE_AND_ONLY_ONCE`**. Set **`QOS_DUPLICATES_OK`** mode, and then check for and discard duplicate messages at the destination.

You can implement **`QOS_ONCE_AND_ONLY_ONCE`** behaviour at the application level by maintaining a cache of received message IDs on disk and comparing received messages to this cache. Because the cache would only be valid for a certain period of time, this approach is not infallible, but can be a useful alternative depending on your application.

8.2. BRIDGE DEPLOYMENT

You can deploy a message bridge by adding a managed bean descriptor into the **`deploy`** directory of the JBoss Enterprise Application Platform installation that contains JBoss Messaging.

8.3. BRIDGE CONFIGURATION

The following code is an example configuration of the message bridge, showing all attributes. Some attributes have been commented out for this configuration, since not all attributes should be specified at once. Those attributes that

Example 8.1. Message Bridge Configuration

```
<mbean code="org.jboss.jms.server.bridge.BridgeService"
  name="jboss.messaging:service=Bridge,name=TestBridge"
  xmbean-dd="xmdesc/Bridge-xmbean.xml">

  <!-- The JMS provider loader that is used to lookup the source
       destination
  -->
  <depends optional-attribute-name="SourceProviderLoader">
  jboss.messaging:service=JMSProviderLoader,name=JMSProvider</depends>

  <!-- The JMS provider loader that is used to lookup the target
       destination
  -->
  <depends optional-attribute-name="TargetProviderLoader">
  jboss.messaging:service=JMSProviderLoader,name=JMSProvider</depends>
  <!-- The JNDI lookup for the source destination -->
  <attribute name="SourceDestinationLookup">/queue/A</attribute>

  <!-- The JNDI lookup for the target destination -->
  <attribute name="TargetDestinationLookup">/queue/B</attribute>

  <!-- The username to use for the source connection
  <attribute name="SourceUsername">bob</attribute>
  -->

  <!-- The password to use for the source connection
  <attribute name="SourcePassword">BobSecur3</attribute>
  -->

  <!-- The username to use for the target connection
  <attribute name="TargetUsername">mary</attribute>
  -->

  <!-- The password to use for the target connection
  <attribute name="TargetPassword">MaryS3cur3</attribute>
  -->

  <!-- Optional: The Quality Of Service mode to use, one of:
    QOS_AT_MOST_ONCE = 0;
    QOS_DUPLICATES_OK = 1;
    QOS_ONCE_AND_ONLY_ONCE = 2;
  -->
  <attribute name="QualityOfServiceMode">0</attribute>

  <!-- JMS selector to use for consuming messages from the source
  <attribute name="Selector">specify jms selector here</attribute>
  -->

  <!-- The maximum number of messages to consume from the source
       before sending to the target
  -->
  <attribute name="MaxBatchSize">5</attribute>

  <!-- The maximum time to wait (in ms) before sending a batch to the
       target even if MaxBatchSize is not exceeded. -1 means wait
forever
  -->
  <attribute name="MaxBatchTime">-1</attribute>

  <!-- If consuming from a durable subscription this is the subscription
       name
       <attribute name="SubName">mysub</attribute>
  -->

  <!-- If consuming from a durable subscription this is the client ID to
       use
       <attribute name="ClientID">myClientID</attribute>
  -->

  <!-- The number of ms to wait between connection retrues in the event
       connections to source or target fail
  -->
  <attribute name="FailureRetryInterval">5000</attribute>

  <!-- The maximum number of connection retries to make in case of
failure,
       before giving up -1 means try forever
  -->
  <attribute name="MaxRetries">-1</attribute>

  <!-- If true then the message ID of the message before bridging will
be
       added as a header to the message so it is available to the
       receiver. Can then be sent as correlation ID to correlate in a
       distributed request-response
  -->
  <attribute name="AddMessageIDInHeader">false</attribute>

    </mbean>
```

Message Bridge Configuration Attributes

SourceProviderLoader, TargetProvider Loader

The **`JMSProviderLoader`** managed bean is used by the bridge to look up the source connection factory and source destination. By default, JBoss Enterprise Application Platform ships with one **`JMSProviderLoader`**, which is deployed in the **`$JBOSS_HOME/server/$PROFILE/deploy/messaging/jms-ds.xml`** file, and serves as the default local **`JMSProviderLoader`**. For a clustered configuration, **`hajndi-jms-ds.xml`** performs the same role.

If your source or target destination is on a different server, or corresponds to a non-JBoss JMS Provider, you can deploy another **`JMSProviderLoader`** managed bean instance that the bridge can use to contact the destination on the remote JMS Provider.

To use **`QOS_ONCE_AND_ONLY_ONCE`** delivery with a remote non-JBoss Messaging source or target, the remote JMS Provider must provide a fully-functional JMS XA resource implementation that works remotely from the server.

SourceDestinationLookup

The full JNDI lookup for the source destination, via the **`SourceProviderLoader`**, such as **`/queue/mySourceQueue`**.

TargetDestinationLookup

The full JNDI lookup for the target destination, via the **`TargetProviderLocator`**, such as **`/topic/myTargetTopic`**.

SourceUsername

An optional attribute that specifies the username used when creating the source connection.

SourcePassword

An optional attribute that specifies the password used when creating the source connection.

TargetUsername

An optional attribute that specifies the username used when creating the target connection.

TargetPassword

An optional attribute that specifies the password used when creating the target connection.

QualityOfServiceMode

An integer representing the desired *quality of service* mode. The possible values are:

- **`0`** to represent **`QOS_AT_MOST_ONCE`**
- **`1`** to represent **`QOS_DUPLICATES_OK`**
- **`2`** to represent **`QOS_ONCE_AND_ONLY_ONCE`**

See Section 8.1, "Message Bridge Overview" for a complete explanation of these modes.

Selector

An optional attribute that lets you provide a JMS selector expression when consuming messages from a source destination. Only messages that match the selector expression are bridged from the source to the target destination. The selector expression must follow the JMS selector syntax, specified here: http://java.sun.com/j2ee/1.4/docs/api/javax/jms/Message.html.

For optimal performance, apply source topic subscription selectors to source queue consumers.

MaxBatchSize

Specifies the maximum number of messages to consume from the source destination before sending a message batch to the target destination. Its value must be greater than or equal to **`1`**.

MaxBatchTime

Specifies the longest period (in milliseconds) to wait before sending a message batch to the target, even if the **`MaxBatchSize`** has not been reached. Its value must be either **`-1`** (wait forever) or greater than or equal to **`1`** to specify a time.

SubName

Represents the name of the durable subscription that will consume from the source destination topic.

ClientID

Represents the JMS client ID to use when creating or looking up the durable subscription that will consume from the source destination topic.

FailureRetryInterval

The period of time (in milliseconds) to wait between attempting to recreate the connection to the source or target server after failure is detected.

MaxRetries

The number of times to attempt to recreate the connection to the source or target server after failure is detected. The bridge will then stop attempting to recreate the connection. A value of **`-1`** means that the bridge will continue to attempt to reconnect forever.

AddMessageIDInHeader

When `true`, the original message ID is added to the `JBossMessage.JBOSS_MESSAGING_BRIDGE_MESSAGE_ID_LIST` header of the message being sent to the destination. If the message is bridged multiple times, each message ID is added to the header. This enables a distributed request-response pattern.

CHAPTER 9. ENABLING JBOSS MESSAGING ORDERING GROUP

This section describes how to use the JBoss Messaging ordering group feature to achieve strict message ordering.

Message ordering groups is the JBoss Messaging implementation of strict message ordering. When the ordering group feature is enabled, message priorities no longer have an influence on the order that the messages are delivered. Messages in a particular ordering group will be delivered in the exact order that they arrive at the target queue (FIFO).

Ordering groups are identified by their string names and obey the following rules:

Rule One

The messages that form a part of an ordering group are delivered one at a time. The next message will not be delivered until the delivery of a previous message is completed. Message delivery completion is signaled by various means, depending on the acknowledge mode settings;

- The **`CLIENT_ACKNOWLEDGE`** mode results in the **`Message.acknowledge()`** method signaling the completion state.
- The **`AUTO_ACKNOWLEDGE`** and **`DUPS_OK_ACKNOWLEDGE`** modes result in the completion of the message being identified by either of the following;
  - a successful return from one of the **`MessageConsumer.receive()`** methods, or
  - a successful return from the **`onMessage()`** call of the **`MessageListener()`**.

NOTE

If the message consumer is closed, the message being processed at the time of its closure will be deemed as completed. This is regardless of whether an **`*_ACKNOWLEGE`** is called prior to the closure of the consumer.

Rule Two

In the case of the transactional receipt of messages, the next message will not be delivered until the transaction has been committed which includes the acknowledgment of the receipt of the current message. If the transaction is rolled back, the message will be canceled, sent back to the JMS server and made available for the next delivery.

9.1. HOW TO ENABLE MESSAGE ORDERING GROUP

JBoss Messaging ordering group may be enabled by one of two means. Either using the API or by making configuration changes.

9.1.1. Enabling with the API

To make use of JBoss Messaging's ordering group feature, it is necessary to create a **`JBossMessageProducer`** as demonstrated in the following code sample:

```
JBossMessageProducer producer=
(JBossMessageProducer)session.createProducer(queue);
```

Once created, JBossMessageProducer provides two methods for starting and ending an ordering group.

enableOrderingGroup()

The following code sample demonstrates how to create an ordering group with the name **`ogrpName`**.

```
public void enableOrderingGroup(String ogrpName) throws JMSException
```

Once called, the producer will send messages on behalf of the ordering group. If a **`null`** parameter is supplied, the name of the ordering group will be automatically generated. Calling this method more than once will override any previous method calls.

disableOrderingGroup()

The following code samples demonstrates how to stop producing ordering group messages.

```
public void disableOrderingGroup() throws JMSException
```

Once called, the producer will stop sending out ordering group messages and resume its default behavior.

9.1.2. Configuration Changes

Users can configure a JBoss Messaging connection factory to enable the ordering group feature. To achieve this, two new attributes are added to the factory service configuration file. These are:

- **`EnableOrderingGroup;`**
  - set this property to **`true`** to enable the ordering group feature. The default value for this property is **`false`**.
- **`DefaultOrderingGroupName;`**
  - the default name for the message ordering group. The group name will be generated automatically if this attribute is missing.

NOTE

Once configured to enable the ordering group feature on a connection factory, all messages that are sent from any producers created from the connection factory become ordering group messages.

The following factory service configuration file sample demonstrates how to enable the ordering group feature:

```
<mbean code="org.jboss.jms.server.connectionfactory.ConnectionFactory";
  name="jboss.messaging.connectionfactory:service=ConnectionFactory";
  xmbean-dd="xmdesc/ConnectionFactory-xmbean.xml">
  <depends optional-attribute-name="ServerPeer">
    jboss.messaging:service=ServerPeer
  </depends>
  <depends optional-attribute-name="Connector">
    jboss.messaging:service=Connector,transport=bisocket
  </depends>
```

```
    <depends>
      jboss.messaging:service=PostOffice
    </depends>

    <attribute name="JNDIBindings">
      <bindings>
      <binding>/MyConnectionFactory</binding>
      <binding>/XAConnectionFactory</binding>
      <binding>java:/MyConnectionFactory</binding>
      <binding>java:/XAConnectionFactory</binding>
      </bindings>
    </attribute>

    <!-- This are the two properties -->
    <attribute name="EnableOrderingGroup">true</attribute>
    <attribute name="DefaultOrderingGroupName">MyOrderingGroup</attribute>
  </mbean>
```

The advantage of enabling the ordering group feature by making configuration changes is the ease with which message ordering functionality can be achieved without the need for code changes.

9.2. NOTES AND LIMITATIONS

The following points should be noted in regard to ordering group functionality:

- Queues must be used with the ordering group feature. The feature will not work with topics.
- The ordering group feature should not be used in conjunction with message selectors and scheduled delivery.
- A message is considered completed, and the next message will be available for delivery, if the original message is dead or has expired. A dead message is moved to the **`DLQ`** whereas an expired message is moved to the **`ExpiryQueue`**.
- When using a **`ConnectionConsumer`**, the ordering of the messages will be observed. However, the **`ConnectionConsumer`** does not control which session will receive the next message.
- In the case of a Distributed Queue, the user should use **`HASingleton`** to ensure that the ordering group feature functions correctly.

APPENDIX A. REVISION HISTORY

Revision 5.1.0-111.400	**2013-10-31**	**Rüdiger Landmann**

Rebuild with publican 4.0.0

Revision 5.1.0-111	**2012-07-18**	**Anthony Towns**

Rebuild for Publican 3.0

Revision 5.1.0-100	**Wed Sep 15 2010**	**Jared Morgan**

Changed version number in line with new versioning requirements.

Revised for JBoss Enterprise Application Platform 5.1.0.GA, including:

Rebased Chapter 6 and 7 for the Common Criteria 5.1 project. Applied code highlighting where appropriate.

JBOSSCC-49 - Incorporated all comments from JBoss Messaging guide rebase.

Revision 5.1.0-100 **Wed Sep 15 2010** **Jared Morgan**

Changed version number in line with new versioning requirements.

Revised for JBoss Enterprise Application Platform 5.1.0.GA, including:

Rebased Chapter 6 and 7 for the Common Criteria 5.1 project. Applied code highlighting where appropriate.

JBOSSCC-49 - Incorporated all comments from JBoss Messaging guide rebase.