JBoss Messaging 1.3 User's Guide

Enterprise Messaging from JBoss

Table of Contents

1

Introducing JBoss Messaging Release 1.3.0.GA

JBoss Messaging is a high performance JMS provider in the JBoss Enterprise Middleware Stack (JEMS). It is a complete rewrite of JBossMQ, the legacy JBoss JMS provider.

JBoss Messaging will be the default JMS provider in later versions of JBoss Enterprise Application Platform, and JBoss Service Integration Platform. It will also be the default JMS provider in JBoss Application Server 5, and is the default JMS provider for JBoss ESB.

JBoss Messaging is an integral part of Red Hat's strategy for messaging.

Compared with JBossMQ, JBoss Messaging offers vastly improved performance in both single node and clustered environments.

Please see this wiki page [http://wiki.jboss.org/wiki/Wiki.jsp?page=JBossMessagingPerformanceResultsPre1_0] for performance benchmarks and ??? on how to generate your own performance benchmarks.

JBoss Messaging also features a much better modular architecture that will allow us to add more features in the future.

JBoss Messaging can be easily installed in JBoss Application Server 4.2 using a few simple steps to remove JBoss MQ and replace with JBoss Messaging. Once JBoss Messaging becomes the default JMS provider in JBoss Application Server 4.2, there will be no need to do any manual installation.

From release 1.3.0.GA onwards JBoss Messaging is designed for JBoss 4.2 only.

The procedure of swapping JMS providers is presented in detail in this manual. In Chapter 5 we discuss how to install and use JBoss Messaging in a JBoss AS 4.2 servers. We cover JBoss Messaging-specific configuration options, as well as how to run the build-in sanity / performance tests.

This guide is work in progress, as new features will be added to the 1.2 baseline at a very quick pace.

Please send your suggestions or comments to the JBoss Messaging user forum [http://www.jboss.org/index.html?module=bb&op=viewforum&f=238].

Permanent Team: Tim Fox - Project Lead, Clebert Suconic - Core Developer, Sergey Koshcheyev - Core Developer

Contributors: Ovidiu Feodorov, Adrian Brock, Alex Fu, Luc Texier, Aaron Walker, Rajdeep Dua, Madhusudhan Konda, Juha Lindfors and Ron Sigal.

Other thanks to Pete Bennett, Jay Howell and David Boeren

2

Introduction

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

JBoss Messaging implements a high performance, robust messaging core that is designed to support the largest and most heavily utilized SOAs, enterprise service buses (ESBs) and other integration needs ranging from the simplest to the highest demand networks.

It will allow you to smoothly distribute your application load across your cluster, intelligently balancing and utilizing each nodes CPU cycles, with no single point of failure, providing a highly scalable and perform-ant clustering implementation.

JBoss Messaging includes a JMS front-end to deliver messaging in a standards-based format as well as being designed to be able to support other messaging protocols in the future.

JBoss Messaging is destined to become an integral part of the JBoss Enterprise Application Platform, and the new Service Integration Platform.

Currently it is available for embedded use within the JBoss Application Server 4.2.0.GA or later (JBossAS). Work to integrate JBoss Messaging with the new JBoss Microcontainer is under way.

JBoss Messaging is also an integral part of Red Hat's strategy for messaging. JBoss Messaging is fully committed to AMQP (AMQP [http://www.amqp.org])- the new messaging standard from Red Hat and others. Later versions of JBoss Messaging will support AMQP, and JBoss Messaging will be focussed on becoming the premier AMQP Java broker.

2.1. JBoss Messaging support cover from Red Hat

JBoss Messaging is destained to become part of both Application Server Platform (JBoss 4.2 series) and Service Integration Platform (JBoss ESB 4 series) as default JMS provider. Production support will then be fully available for these plaforms and it will cover JBoss Messaging.

Currently developer support is available for JBoss Messaging when installed in JBoss 4.2.x

2.2. JBoss Messaging Features

JBoss Messaging provides:

- A fully compatible and Sun certified JMS 1.1 implementation, that currently works with a standard 4.2.x JBoss Application Server installation.

- 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.

Other JBoss Messaging features include:

- Publish-subscribe and point-to-point messaging models
- Persistent and non-persistent messages
- Guaranteed message delivery that ensures that messages arrive once and only once where required
- Transactional and reliable - supporting ACID semantics
- Customizable security framework based on JAAS
- Fully integrated with JBoss Transactions (formerly known as Arjuna JTA) for full transaction recoverability.
- Extensive JMX management interface
- Support for most major databases including Oracle, Sybase, MS SQL Server, PostgreSQL and MySQL
- HTTP transport
- SSL transport

Clustering features:

- Distributed queues. Messages sent to a distributed queue while attached to a particular node will be routed to a queue instance on a particular node according to a routing policy.
- Distributed topics. Messages sent to a distributed topic while attached at a particular node will be received by subscriptions on other nodes.
- Fully reliable message distribution. Once and only once delivery is fully guaranteed. When sending messages to a topic with multiple durable subscriptions across a cluster we guarantee that message reaches all the subscriptions (or none of them in case of failure).
- Pluggable routing implementation. The policy for routing messages to a queue is fully pluggable and easily replaceable. The default policy always chooses a queue at the local node if there is one, and if not, it round robins between queues on different nodes.
- Intelligent message redistribution policy. Messages are automatically distributed between nodes depending on how fast or slow consumers are on certain nodes. If there are no or slow consumers on a particular queue node, messages will be pulled from that queue to a queue with faster consumers on a different node. The policy is fully pluggable.
- Shared durable subscriptions. Consumers can connect to the same durable subscription while attached to different nodes. This allows processing load from durable subscriptions to be distributed across the cluster in a similar way to queues.

- High availability and seamless fail-over. If the node you are connected to fails, you will automatically fail over to another node and will not lose any persistent messages. You can carry on with your session seamlessly where you left off. Once and only once delivery of persistent messages is respected at all times.
- Message bridge. JBoss Messaging contains a message bridge component which enables you to bridge messages between any two JMS1.1 destinations on the same or physical separate locations. (E.g. separated by a WAN)

2.3. Compatibility with JBossMQ

JBoss MQ is the JMS implementation currently shipped within JBoss AS. Since JBoss Messaging is JMS 1.1 and JMS 1.0.2b compatible, the JMS code written against JBossMQ will run with JBoss Messaging without any changes.

JBoss Messaging does not have wire format compatibility with JBoss MQ so it would be necessary to upgrade JBoss MQ clients with JBoss Messaging client jars

Important

Even if JBoss Messaging deployment descriptors are very similar to JBoss MQ deployment descriptors, they are *not* identical, so they will require some simple adjustments to get them to work with JBoss Messaging. Also, the database data model is completely different, so don't attempt to use JBoss Messaging with a JBoss MQ data schema and vice-versa.

3

JBoss Messaging Clustering

This section of the userguide gives a brief overview of the features available in JBoss Messaging. It gives a high level explanation of how clustering works.

3.1. JBoss Messaging Clustering Overview

Here's a brief overview of how clustering works in JBoss Messaging

Clustered destinations (queues and topics) can be deployed at all or none of the nodes of the cluster. A JMS client uses HA JNDI to lookup the connection factory. When creating connections using that connection factory a client side load balancing policy will automatically chose a node to connect to.

The JMS client has now made a connection to a node where it can create sessions, message producers and message consumers and browsers and send or consume messages, using the standard JMS api. When a distributed queue is deployed across the cluster, individual partial queues are deployed on each node.

When a message is sent from a message producer attached to a particular node to a distributed queue, a routing policy determines which partial queue will receive the message. By default the router will always pass the message to a local queue, if there is one, this is so we avoid unnecessary network traffic. If there is no local queue then a partial queue on a different node will be chosen by the router, by default this will be round robin between remote partial queues.

When a message is sent to a distributed topic while attached to a node, there may be multiple subscriptions on different nodes that need to receive the message. Depending on the number and location of subscriptions, the message may be multicast or unicast across the cluster so the other nodes can pick it up. (All group communication, unicast, multicast and group management is handled by JGroups.)

In the case of shared durable subscriptions, if a durable subscription with the same name exists on more than node, then only one of the instances needs to receive the message. Which one is determined by the same routing policy used to route messages to partial queues. All of this is accomplished without losing the reliability guarantees required by JMS.

Subscriptions (both durable and non durable) can be created on all nodes and will receive messages sent via any node. What happens if the consumers on one queue/subscription are faster/slower than consumers on another? Normally this would result in messages building up on that queue and fast consumers being starved of work on another, thus wasting CPU cycles on the node that could be put to good use. The most degenerate example is of a queue containing many messages then the consumers being closed on that queue. The messages might potentially remain stranded on the queue until another consumer attaches. A routing policy is no use here, since the messages have already been routed to the queuee and the consumers closed / slowed down after they were routed there. JBoss Messaging can deal with this problem by intelligently pulling messages from other less busy nodes, if it detects idle consumers on the fast node and slow consumers on another node.

3.2. Clustering Architectural Overview

One of the fundamental Messaging Core building blocks is the "Post Office". A JBoss Messaging Post Office is message routing component, which accepts messages for delivery and synchronously forwards them to their destination queues or topic subscriptions.

There is a single Post Office instance per JBoss Messaging server (cluster node). Both queues and topics deployed on a JBoss Messaging node are "plugged" into that Post Office instance. Internally JBoss Messaging only deals with the concepts of queues, and considers a topic to just be a set of queues (one for each subscription). Depending on the type of subscription - durable or non-durable - the corresponding queue saves messages to persistent storage or it just holds messages in memory and discards them when the non-durable subscription is closed.

Therefore, for a JMS queue, the Post Office routes messages to one and only one core queue, depending on the queue name, whereas for a JMS topic, the Post Office routes a message to a set of core queues, one for each topic subscription, depending on the topic name.

Clustering across multiple nodes is achieved by clustering Post Office instances. Each JBoss Messaging cluster node runs a Clustered Post Office instance, which is aware of the presence of all other clustered Post Offices in the cluster. There is an one-to-one relationship between cluster nodes and clustered Post Office instances. So, naturally, the most important piece of clustering configuration is the *clustered Post Office service configuration*, covered in detail below.

Clustered Post Office instances connect to each other via JGroups and they heavily rely on JGroups group management and notification mechanisms. *JGroups stack configuration* is an essential part of JBoss Messaging clustering configuration. JGroups configuration is only briefly addressed in this guide. Detailed information on JGroups can be found in JGroups release documentation or on-line at http://www.jgroups.org or http://wiki.jboss.org/wiki/Wiki.jsp?page=JGroups

When routing messages, a clustered Post Office has a choice of forwarding the message to local queues or remote queues, plugged into remote Post Office instances that are part of the same cluster. Local queues are usually preferred, but if a local queue is part of a distributed queue, has no consumers, and other local queues part of the same distributed queue have consumers, messages can be automatically redistributed, subject of the message redistribution policy in effect. This allows us to create distributed queues and distributed topics. *Message redistribution configuration* is another subject that we will insist on.

4

Download Software

The official JBoss Messaging project page is http://labs.jboss.com/jbossmessaging/ [http://labs.jboss.com/jbossmessaging].

The download location is the JBoss Labs Messaging Project download zone: http://labs.jboss.com/jbossmessaging/

4.1. The JBoss Messaging Release Bundle

The JBoss Messaging release bundle (`jboss-messaging-1.3.x.zip`) will expand in a `jboss-messaging-1.3.x` directory that contains:

- `jboss-messaging.sar` - the service archive that contains JBoss Messaging

  Warning

  Do not simply attempt to copy the archive under a JBoss instance `deploy` directory, since additional steps (such as un-installing JBossMQ and various other configurations tasks) are required for a successful installation. See Chapter 5 for more details.

- `jboss-messaging-client.jar` - the client-side library that need to be in the classpath of the client that opens a remote connection to the Messaging server.
- `util` - a collection of `ant` configuration files used to automate installation. See Chapter 5 for more details.
- `examples` - a collection of examples that should run out of the box and help you validate the installation. Detailed instructions are provided with each example, which range from very simple JMS queue and topic examples to EJB 2.1, EJB3 and clustering examples. The `examples/config` sub-directory contains various configuration file examples.
- `docs` - this user's guide.
- `test-results` - the output of the functional testsuite, stress and smoke test runs for this release. All these files have been generated during the release procedure and are bundled here for reference.
- `api` - the Messaging API javadoc.
- `README.html` - The release intro document that contains pointers to various other resources, including this Guide.

4.2. SVN Access

If you want to experiment with the latest developments you may checkout the latest code from the Messaging SVN trunk. Be aware that the information provided in this manual might then not be accurate. For the latest instructions on how to check out and build source code, please go to Messaging Development wiki page [http://wiki.jboss.org/wiki/Wiki.jsp?page=JBossMessagingDevelopment], specifically "Building and Running JBoss Messaging" [http://wiki.jboss.org/wiki/Wiki.jsp?page=JBossMessagingBuildInstructions] section.

5

JBoss Messaging Installation

This section describes procedures on how to install JBoss Messaging into JBoss AS. At the end of this procedure, you will create a JBoss Messaging configuration that will start a clustered or non-clustered messaging server.

By default, JBoss AS 4.2.0 instance ships with JBossMQ as default JMS provider. In order to use the JBoss AS instance with JBoss Messaging, you need to perform the installation procedure described below.

> **Note**
>
> A JBossMQ and a JBoss Messaging instance cannot coexist, at least not unless special precautions are taken. Do not simply attempt to copy the Messaging release artifact `jboss-messaging.sar` over to the JBoss instance `deploy` directory. Follow one of the alternate installation procedures outlined below instead.

> **Note**
>
> We only recommend and support installing JBoss Messaging in JBoss AS 4.2 or later. You should avoid using JBoss Messaging on any version of JBoss AS prior to JBoss 4.2.0.GA, such as 4.0.5.GA and 4.0.4.GA.

5.1. Installing JBoss Messaging on JBoss AS 4.2

In this section we present five different methods of installing JBoss Messaging in JBoss AS 4.2

- If you do not want clustering functionality and have a completely clean JBoss AS 4.2.0 installation then you can do an Section 5.1.1.
- If you do not want clustering functionality and have a JBoss 4.2.0 that you have changed in some way from a clean JBoss AS 4.2.0 installation then you will need to do a Section 5.1.2.
- If you want clustering functionality and have a completely clean JBoss AS 4.2.0 installation then you can do an Section 5.1.3.
- If you want clustering functionality and have a JBoss 4.2.0 that you have changed in some way from a clean JBoss AS 4.2.0 installation then you will need to do a Section 5.1.4.
- > **Warning**
  >
  > Not recommended!

  If you to install in JBoss 4.0.x and are willing to cope with any jar incompatibilities that may arise then you will need to perform a Section 5.2.

5.1.1. Automated Non-clustered Installation from clean JBoss AS 4.2 installation

Note

This procedure should only be performed from a clean JBoss AS 4.2 installation. If you have modifed the JBoss AS 4.2 installation at all since installation then you will need to perform a manual JBoss Messaging installation

Set up the `JBOSS_HOME` environment variable to point to the JBoss 4.2 installation you want to use JBoss Messaging with. Run the installation script, available in the `util` directory of the release bundle.

```
cd util
ant -f release-admin.xml
```

The installation script will create a `$JBOSS_HOME/server/messaging` configuration.

Warning

For use in production it is highly recommended to use a different database other than the HSQLDB. HSQLDB does not have any transaction isolation. For simple non clustered development evaluation HSQLDB may suffice. If you decide to use a different database please follow the follow steps:

- Replace `JBOSS_CONFIG/deploy/jboss-messaging.sar/hsqldb-persistence-service.xml` by the persistence-service of from `<downloadPackage>/examples/config`.
- Configure a JCA datasource using an example from `$JBOSS_HOME/docs/examples/jca` and copying to `$JBOSS_CONFIG/deploy`

  JBoss Messaging uses `DefaultDS` by default so you should configure your datasource to bind to that
- Remove hsqldb-ds.xml from `$JBOSS_CONFIG/deploy`
- Copy your database driver to `$JBOSS_CONFIG/lib`

  Your database driver can probably be downloaded from your database providers web site

There are few extra steps at Section 5.1.5

That's it!

Note

If you want to create a JBoss Messaging configuration with a different name, you can specify the `messaging.config.name` system property when you run the command:

```
cd util
ant -f release-admin.xml -Dmessaging.config.name=mymessaging
```

5.1.2. Manual Non-clustered Installation

Note

This installation procedure should be performed if you are installing into a JBoss AS configuration that you have changed in some way from the default JBoss AS distribution. If you are just using the standard, untouched JBoss AS 4.2 distribution you can use the automated procedure above

For this procedure we assume you already have your custom configuration located at `JBOSS_CONFIG=$JBOSS_HOME/server/<myconfiguration>`, and that it contains a JBoss MQ installation.

You don't actually have to create an environment variable `JBOSS_CONFIG`, this is just used in the installation instructions to describe the steps

- Move `$JBOSS_CONFIG/deploy/jms/jms-ds.xml` and `$JBOSS_CONFIG/deploy/jms/jms-ra.rar` to `$JBOSS_CONFIG/deploy`
- Remove the old JBoss MQ by removing the directory `$JBOSS_CONFIG/deploy/jms.`
  .

  Remove the old JBoss MQ jar file: `$JBOSS_CONFIG/lib/jbossmq.jar`
- Add a security policy called "messaging" on $JBOSS_CONFIG/config/login-config.xml. You could use this as an example, or create one according to JBoss Security Documentation:

```
<application-policy name = "messaging">
<authentication>
<login-module code = "org.jboss.security.auth.spi.UsersRolesLoginModule" flag = "required" >
   <module-option name = "unauthenticatedIdentity">guest</module-option>
   <module-option name = "usersProperties">props/messaging-users.properties</module-option>
   <module-option name = "rolesProperties">props/messaging-roles.properties</module-option>
</login-module>
</authentication>
</application-policy>
```

In case you are using the above policy you should also create files `messaging-users.properties` and `messaging-roles.properties` in the `$JBOSS_CONFIG/config/props/` directory
.

Note

You can find an example `messaging-users.properties` and `messaging-roles.properties` in the JBoss Messaging distribution in the `<downloadPackage>src/etc/server/default/config` directory.

```
# messaging-roles.properties
# Add roles as you like
# user=role1,role2,...
#
```

```
guest=guest
```

```

# messaging-users.properties

# Add users as you like
# user=password
#
guest=guest
```

- Unzip jboss-messaging.sar from your download package into the directory `JBOSS_CONFIG/deploy/jboss-messaging.sar`

  JBoss Messaging sar should be deployed unzipped (exploded) so you have easy access to its config files which are stored there.

- Copy jboss-messaging.jar from your download package into the directory `JBOSS_CONFIG/lib`

  jboss-messaging.jar needs to go in the lib directory so it is accessible to other services e.g. the JBoss Transactions Recovery Manager

- **Warning**

  For use in production it is highly recommended to use a different database other than the HSQLDB. HSQLDB does not have any transaction isolation. For simple non clustered development evaluation HSQLDB may suffice. If you decide to use a different database please follow the follow steps:

  - Replace `JBOSS_CONFIG/deploy/jboss-messaging.sar/hsqldb-persistence-service.xml` by the persistence-service of from `<downloadPackage>/examples/config`.

  - Configure a JCA datasource using an example from `$JBOSS_HOME/docs/examples/jca` and copying to `$JBOSS_CONFIG/deploy`

    JBoss Messaging uses `DefaultDS` by default so you should configure your datasource to bind to that

  - Remove hsqldb-ds.xml from `$JBOSS_CONFIG/deploy`

  - Copy your database driver to `$JBOSS_CONFIG/lib`

    Your database driver can probably be downloaded from your database providers web site

- There are few extra steps at Section 5.1.5

- That's it!

5.1.3. Clustered Automated Installation

Note

> This procedure should only be performed from a clean JBoss AS 4.2 installation. If you have modifed the JBoss AS 4.2 installation at all since installation then you will need to perform a manual clustered JBoss Messaging installation

For the rest of the procedure we assume JBOSS_CONFIG refers to your new messaging configuration (e.g. messaging-node0)

You don't actually have to create an environment variable `JBOSS_CONFIG`, this is just used in the installation instructions to describe the steps

- Set up the `JBOSS_HOME` environment variable to point to the JBoss 4.2 installation you want to use JBoss Messaging with. Run the installation script, available in the `util` directory of the release bundle as follows:

```
cd util
ant -f release-admin.xml -Dmessaging.config.source=all -Dmessaging.config.name=messaging-node0
```

  In the above you would substitute `messaging-node0` with whatever is the name you want to give the configuration. If you will have several cluster nodes on the same machine, e.g. for development purposes then a good convention is to name them `messaging-node0, messaging-node1` to match `messaging-node<ServerPeerID>`

  The installation script will create a `$JBOSS_HOME/server/messaging-node0` configuration. (If you have chosen `messaging-node0`)

-
  ### Warning

  For a clustered installation it is mandatory that a shared database is available to all nodes in the cluster. The default JBoss AS uses HSQLDB for its database which is a local shared database. Therefore in order to use clustering you must replace this with a different shared database. If the database is not replaced then clustering will not work.

  - Replace `$JBOSS_CONFIG/deploy/jboss-messaging.sar/hsqldb-persistence-service.xml` by the `clustered-<databasename>-persistence-service` from `<downloadPackage>/examples/config`. For instance `clustered-mysql-persistence-service.xml`
  - Configure a JCA datasource using an example from `$JBOSS_HOME/docs/examples/jca` and copying to `$JBOSS_CONFIG/deploy`

    JBoss Messaging uses `DefaultDS` by default so you should configure your datasource to bind to that
  - Remove hsqldb-ds.xml from `$JBOSS_CONFIG/deploy`
  - Copy your database driver to `$JBOSS_CONFIG/lib`

    Your database driver can probably be downloaded from your database provider's web site

- Ensure the `ServerPeerID` MBean constructor parameter in messaging-service.xml is unique for each node on the cluster. The `ServerPeerID` value must be a valid integer.

Warning

Each node must have a unique `ServerPeerID` for clustering to work!

- If you want to run multiple JBoss Messaging nodes on the same box using the same IP address, e.g. for development purposes, then you can use the ServiceBindingManager to do this as follows:

  - Uncomment binding manager service from $JBOSS_CONFIG/conf/jboss-service.xml
  - Specify the desired port rage (e.g. ports-01, ports-02... etc)
  - Look at $JBOSS_HOME/docs/examples/binding-manager/sample-bindings.xml. On each port range, JBoss Remoting configuration should look like:

```
<service-config name="jboss.messaging:service=Connector,transport=bisocket"
                delegateClass="org.jboss.services.binding.AttributeMappingDelegate">
   <delegate-config>
      <attribute name="Configuration"><![CDATA[
     <config>
      <invoker transport="bisocket">
         <attribute name="marshaller" isParam="true">org.jboss.jms.wireformat.JMSWireFormat</attrib
         <attribute name="unmarshaller" isParam="true">org.jboss.jms.wireformat.JMSWireFormat</attr
         <attribute name="dataType" isParam="true">jms</attribute>
         <attribute name="socket.check_connection" isParam="true">false</attribute>
         <attribute name="timeout" isParam="true">0</attribute>
         <attribute name="serverBindAddress">${jboss.bind.address}</attribute>
         <attribute name="serverBindPort">4657</attribute>
         <attribute name="leasePeriod">10000</attribute>
         <attribute name="clientSocketClass" isParam="true">org.jboss.jms.client.remoting.ClientSoc
         <attribute name="serverSocketClass">org.jboss.jms.server.remoting.ServerSocketWrapper</att
         <attribute name="numberOfRetries" isParam="true">1</attribute>
         <attribute name="numberOfCallRetries" isParam="true">1</attribute>
         <attribute name="clientMaxPoolSize" isParam="true">50</attribute>
      </invoker>
      <handlers>
         <handler subsystem="JMS">org.jboss.jms.server.remoting.JMSServerInvocationHandler</handler
      </handlers>
   </config>
      ]]></attribute>
   </delegate-config>
   <binding port="4657"/>
</service-config>
```

Warning

You must ensure that the config (like above) is identical to that in `remoting-service.xml` With the exception of the actual serverBindPort which clearly must be different for each ports range. Please note that the default JBoss Messaging service binding manager bindings in `sample-bindings.xml` shipped with JBAS 4.2.0 is out of date and you will need to copy the config from `remoting-service.xml`

You should ensure that each node is configured to use a different ports range.

- There are few extra steps at Section 5.1.5

- That's it

5.1.4. Clustered Manual Installation

Note

This installation procedure should be performed if you are installing into a JBoss AS configuration that you have changed in some way from the default JBoss AS distribution. If you are just using the standard, untouched JBoss AS 4.2 distribution you can use the automated procedure above

For this procedure we assume you already have your custom configuration located at `JBOSS_CONFIG=$JBOSS_HOME/server/<myconfiguration>`, and that it contains a JBoss MQ installation.

You don't actually have to create an environment variable `JBOSS_CONFIG`, this is just used in the installation instructions to describe the steps

- Move `$JBOSS_CONFIG/deploy/jms/hajndi-jms-ds.xml` and `$JBOSS_CONFIG/deploy/jms/jms-ra.rar` to `$JBOSS_CONFIG/deploy`
- Remove the old JBoss MQ by removing the directory `$JBOSS_CONFIG/deploy/jms`.

  .

  Remove the old JBoss MQ jar file: `$JBOSS_CONFIG/lib/jbossmq.jar`

  Make sure you don't have any JBossMQ files under `$JBOSS_CONFIG/deploy-hasingleton`. For that just remove `$JBOSS_CONFIG/deploy-hasingleton/jms`
- Add a security policy called "messaging" on $JBOSS_CONFIG/config/login-config.xml. You could use this as an example, or create one according to JBoss Security Documentation:

```
<application-policy name = "messaging">
<authentication>
<login-module code = "org.jboss.security.auth.spi.UsersRolesLoginModule" flag = "required" >
   <module-option name = "unauthenticatedIdentity">guest</module-option>
   <module-option name = "usersProperties">props/messaging-users.properties</module-option>
   <module-option name = "rolesProperties">props/messaging-roles.properties</module-option>
</login-module>
</authentication>
</application-policy>
```

In case you are using the above policy you should also create files `messaging-users.properties` and `messaging-roles.properties` in the `$JBOSS_CONFIG/config/props/` directory

.

Note

You can find an example `messaging-users.properties` and `messaging-roles.properties` in the JBoss Messaging distribution in the `<downloadPackage>src/etc/server/default/config` directory.

```
```

```
# messaging-roles.properties
# Add roles as you like
# user=role1,role2,...
#
guest=guest
```

```

# messaging-users.properties

# Add users as you like
# user=password
#
guest=guest
```

- Unzip jboss-messaging.sar from your download package into the directory `JBOSS_CONFIG/deploy/jboss-messaging.sar`

  JBoss Messaging should be deployed unzipped (exploded) so you have easy access to its config files which are stored there.

- Copy jboss-messaging.jar from your download package into the directory `JBOSS_CONFIG/lib`

  jboss-messaging.jar needs to go in the lib directory so it is accessible to other services e.g. the JBoss Transactions Recovery Manager

- ### Warning

  For a clustered installation it is mandatory that a shared database is available to all nodes in the cluster. The default JBoss AS uses HSQLDB for its database which is a local shared database. Therefore in order to use clustering you must replace this with a different shared database. If the database is not replaced then clustering will not work.

  - Replace `$JBOSS_CONFIG/deploy/jboss-messaging.sar/hsqldb-persistence-service.xml` by the `clustered-<databasename>-persistence-service` from `<downloadPackage>/examples/config`.. For instance `clustered-mysql-persistence-service.xml`

  - Configure a JCA datasource using an example from `$JBOSS_HOME/docs/examples/jca` and copying to `$JBOSS_CONFIG/deploy`

    JBoss Messaging uses `DefaultDS` by default so you should configure your datasource to bind to that

  - Remove hsqldb-ds.xml from `$JBOSS_CONFIG/deploy`

  - Copy your database driver to `$JBOSS_CONFIG/lib`

    Your database driver can probably be downloaded from your database provider's web site

- Ensure the `ServerPeerID` MBean constructor parameter in messaging-service.xml is unique for each node on the cluster. The `ServerPeerID` value must be a valid integer.

Warning

Each node must have a unique `ServerPeerID` for clustering to work!

- If you want to run multiple JBoss Messaging nodes on the same box using the same IP address, e.g. for development purposes, then you can use the ServiceBindingManager to do this as follows:

  - Uncomment binding manager service from $JBOSS_CONFIG/conf/jboss-service.xml
  - Specify the desired port rage (e.g. ports-01, ports-02... etc)
  - Look at $JBOSS_HOME/docs/examples/binding-manager/sample-bindings.xml. On each port range, JBoss Remoting configuration should look like:

```
<service-config name="jboss.messaging:service=Connector,transport=bisocket"
                delegateClass="org.jboss.services.binding.AttributeMappingDelegate">
   <delegate-config>
      <attribute name="Configuration"><![CDATA[
     <config>
      <invoker transport="bisocket">
         <attribute name="marshaller" isParam="true">org.jboss.jms.wireformat.JMSWireFormat</attrib
         <attribute name="unmarshaller" isParam="true">org.jboss.jms.wireformat.JMSWireFormat</attr
         <attribute name="dataType" isParam="true">jms</attribute>
         <attribute name="socket.check_connection" isParam="true">false</attribute>
         <attribute name="timeout" isParam="true">0</attribute>
         <attribute name="serverBindAddress">${jboss.bind.address}</attribute>
         <attribute name="serverBindPort">4657</attribute>
         <attribute name="leasePeriod">10000</attribute>
         <attribute name="clientSocketClass" isParam="true">org.jboss.jms.client.remoting.ClientSoc
         <attribute name="serverSocketClass">org.jboss.jms.server.remoting.ServerSocketWrapper</att
         <attribute name="numberOfRetries" isParam="true">1</attribute>
         <attribute name="numberOfCallRetries" isParam="true">1</attribute>
         <attribute name="clientMaxPoolSize" isParam="true">50</attribute>
      </invoker>
      <handlers>
         <handler subsystem="JMS">org.jboss.jms.server.remoting.JMSServerInvocationHandler</handler
      </handlers>
   </config>
      ]]></attribute>
   </delegate-config>
   <binding port="4657"/>
</service-config>
```

Warning

You must ensure that the config (like above) is identical to that in `remoting-service.xml` With the exception of the actual serverBindPort which clearly must be different for each ports range. Please note that the default JBoss Messaging service binding manager bindings in `sample-bindings.xml` shipped with JBAS 4.2.0 is out of date and you will need to copy the config from `remoting-service.xml`

You should ensure that each node is configured to use a different ports range.

- There are few extra steps at Section 5.1.5

- That's it

5.1.5. Extra steps to complete your installation

You should also make these changes on any configuration you choose, to remove all references to the old JBoss-MQ:

- Edit `$JBOSS_CONFIG/jms-ds.xml` and replace jboss.mq by jboss.messaging on every occurrence

  If you are in a clustered installation, then do the above with the file `$JBOSS_CONFIG/hajndi-jms-ds.xml`

- Edit `$JBOSS_CONFIG/config/standardjboss.xml` and set `CreateJBossMQDestination` to false on every occurrence

  Make sure it looks like this:

  `<CreateJBossMQDestination>false</CreateJBossMQDestination>`

  Those Proxies will try to create a Destination on JBossMQ if they can't find it on JNDI, what would cause some errors related to JBoss MQ.

- Edit $JBOSS_CONFIG/config/jboss-service.xml and remove the reference to JBoss MQ on JSR-77 Management Bean:

```
 <!-- ==================================================================== -->
 <!-- JSR-77 Single JBoss Server Management Domain                         -->
 <!-- ==================================================================== -->
 <mbean code="org.jboss.management.j2ee.LocalJBossServerDomain"

  ... Remove this line ...
 <attribute name="JMSService">jboss.mq:service=DestinationManager</attribute>
```

- Change `$JBOSS_CONFIG/conf/login-config.xml` and remove jboss-mq security policies

```
### Remove these lines:

<!-- Security domain for JBossMQ -->
<application-policy name = "jbossmq">
 <authentication>
    <login-module code = "org.jboss.security.auth.spi.DatabaseServerLoginModule"
       flag = "required">
       <module-option name = "unauthenticatedIdentity">guest</module-option>
       <module-option name = "dsJndiName">java:/DefaultDS</module-option>
       <module-option name = "principalsQuery">
             SELECT PASSWD FROM JMS_USERS WHERE USERID=?</module-option>
       <module-option name = "rolesQuery">
             SELECT ROLEID, 'Roles' FROM JMS_ROLES WHERE USERID=?</module-option>
    </login-module>
 </authentication>
</application-policy>

<!-- Security domain for JBossMQ when using file-state-service.xml
<application-policy name = "jbossmq">
 <authentication>
```

```
    <login-module code = "org.jboss.mq.sm.file.DynamicLoginModule"
       flag = "required">
       <module-option name = "unauthenticatedIdentity">guest</module-option>
       <module-option name = "sm.objectname">jboss.mq:service=StateManager</module-option>
    </login-module>
 </authentication>
</application-policy>
-->
```

5.2. JBoss Messaging on JBoss 4.0.x

Warning

You should avoid using JBossMessaging on any version prior to JBoss 4.2.0.GA, such as 4.0.5.GA and 4.0.4.GA. JBoss Messaging uses newer versions of JBoss AOP, JBoss Remoting and JGroups compared to JBoss 4.0.X. If you really need to install JBoss Messaging on those versions you will have to update those jars as described on this section but this might cause problems with other components such as JBoss Web Services and EJB3, and may cause other parts of JBoss AS to stop functioning. Also there may be support implications when running in this configuration since the installation procedure requires overwriting some of the jars in JBoss AS. Do this at your own peril!

- Install JBoss Messaging using the most convenient way described on the previous section using the default configuration as your base (even for a clustered JBoss Messaging)
- Replace the jars on this following list. You can download these jars the repository links provided below:
  - $JBOSS_HOME/server/<SERVER_NAME>/deploy/jboss-aop.deployer/jboss-aop.jar

    JBoss AOP 1.5.5.GA+

    http://repository.jboss.com/jboss/aop/1.5.5.GA/lib/

    (For AOP, sometimes you have to use a specific JAR according to your JVM of choice. Use the most convenient for you)
  - $JBOSS_HOME/server/<SERVER_NAME>/lib/javassist.jar

    Javassist 3.5.0.GA-brew+

    http://repository.jboss.com/javassist/3.5.0.GA-brew/lib/
  - $JBOSS_HOME/server/<SERVER_NAME>/lib/jboss-remoting.jar

    JBoss Remoting 2.2.0.SP4+

    http://repository.jboss.com/jboss/remoting/2.2.0.SP4/lib/
  - $JBOSS_HOME/server/<SERVER_NAME>/lib/jgroups.jar (if using a clustered configuration)

JGroups 2.4.1.SP3-brew+

http://repository.jboss.com/jgroups/2.4.1.SP3-brew/lib/

5.3. Starting the Server

To run the server, execute the `run.bat` or `run.sh` script as appropriate for your operating system, in the `$JBOSS_HOME/bin` directory.

```
cd $JBOSS_HOME/bin
./run.sh -c <config name>
```

Where config_name is the name of the JBoss AS configuration where you have installed messaging. (The default is 'messaging')

A successful JBoss Messaging deployment generates logging output similar to for a non clustered installation (for a clustered installation you will also see extra cluster related output)

```
....
13:19:14,914 WARN  [JDBCPersistenceManager]

JBoss Messaging Warning: DataSource connection transaction isolation should be READ_COMMITTED, but it is
                         Using an isolation level less strict than READ_COMMITTED may lead to data consis
                         Using an isolation level more strict than READ_COMMITTED may lead to deadlock.

13:19:15,166 INFO  [ServerPeer] JBoss Messaging 1.3.0.GA server [0] started
13:19:15,411 INFO  [ConnectionFactory] Connector bisocket://127.0.0.1:4457 has leasing enabled, lease per
13:19:15,412 INFO  [ConnectionFactory] [/ConnectionFactory, /XAConnectionFactory, java:/ConnectionFactory
13:19:15,412 WARN  [ConnectionFactoryJNDIMapper] supportsFailover attribute is true on connection factory
13:19:15,413 WARN  [ConnectionFactoryJNDIMapper] supportsLoadBalancing attribute is true on connection fa
13:19:15,449 INFO  [ConnectionFactory] Connector bisocket://127.0.0.1:4457 has leasing enabled, lease per
13:19:15,449 INFO  [ConnectionFactory] [/ClusteredConnectionFactory, /ClusteredXAConnectionFactory, java:
13:19:15,468 INFO  [QueueService] Queue[/queue/DLQ] started, fullSize=200000, pageSize=2000, downCacheSiz
13:19:15,474 INFO  [QueueService] Queue[/queue/ExpiryQueue] started, fullSize=200000, pageSize=2000, down
13:19:15,476 INFO  [TopicService] Topic[/topic/testTopic] started, fullSize=200000, pageSize=2000, downCa
13:19:15,478 INFO  [TopicService] Topic[/topic/securedTopic] started, fullSize=200000, pageSize=2000, dow
13:19:15,479 INFO  [TopicService] Topic[/topic/testDurableTopic] started, fullSize=200000, pageSize=2000,
13:19:15,482 INFO  [QueueService] Queue[/queue/testQueue] started, fullSize=200000, pageSize=2000, downCa
13:19:15,483 INFO  [QueueService] Queue[/queue/A] started, fullSize=200000, pageSize=2000, downCacheSize=
13:19:15,485 INFO  [QueueService] Queue[/queue/B] started, fullSize=200000, pageSize=2000, downCacheSize=
13:19:15,487 INFO  [QueueService] Queue[/queue/C] started, fullSize=200000, pageSize=2000, downCacheSize=
13:19:15,489 INFO  [QueueService] Queue[/queue/D] started, fullSize=200000, pageSize=2000, downCacheSize=
13:19:15,490 INFO  [QueueService] Queue[/queue/ex] started, fullSize=200000, pageSize=2000, downCacheSize
13:19:15,501 INFO  [QueueService] Queue[/queue/PrivateDLQ] started, fullSize=200000, pageSize=2000, downC
13:19:15,503 INFO  [QueueService] Queue[/queue/PrivateExpiryQueue] started, fullSize=200000, pageSize=200
13:19:15,507 INFO  [QueueService] Queue[/queue/QueueWithOwnDLQAndExpiryQueue] started, fullSize=200000, p
13:19:15,508 INFO  [TopicService] Topic[/topic/TopicWithOwnDLQAndExpiryQueue] started, fullSize=200000, p
13:19:15,511 INFO  [QueueService] Queue[/queue/QueueWithOwnRedeliveryDelay] started, fullSize=200000, pag
13:19:15,512 INFO  [TopicService] Topic[/topic/TopicWithOwnRedeliveryDelay] started, fullSize=200000, pag
13:19:15,514 INFO  [QueueService] Queue[/queue/testDistributedQueue] started, fullSize=200000, pageSize=2
13:19:15,519 INFO  [TopicService] Topic[/topic/testDistributedTopic] started, fullSize=200000, pageSize=2
13:19:15,809 INFO  [ConnectionFactoryBindingService] Bound ConnectionManager 'jboss.jca:service=Connectio
13:19:15,834 INFO  [TomcatDeployer] deploy, ctxPath=/jmx-console, warUrl=.../deploy/jmx-console.war/
13:19:16,322 INFO  [Http11Protocol] Starting Coyote HTTP/1.1 on http-127.0.0.1-8080
```

```
13:19:16,342 INFO  [AjpProtocol] Starting Coyote AJP/1.3 on ajp-127.0.0.1-8009
13:19:16,480 INFO  [Server] JBoss (MX MicroKernel) [4.2.0.GA (build: SVNTag=JBoss_4_2_0_GA date=200705111
```

Note

The warning message `"DataSource connection transaction isolation should be READ_COMMITTED, but it is currently NONE"` is there to remind you that by default JBossAS ships with Hypersonic, an in-memory Java-based database engine, which is apropriate for demo purposes, but not for heavy load production environments. The Critique of Hypersonic [http://wiki.jboss.org/wiki/Wiki.jsp?page=ConfigJBossMQDB] wiki page outlines some of the well-known issues occuring when using this database.

Warning

Before using Messaging in production, you *must* configure the Messaging instance to use an enterprise-class database backend such as MySQL or Oracle, otherwise you risk losing your data. See Section 7.2 for details about replacing Hypersonic.

5.4. Installation Validation

The release bundle contains a series of examples that should run "out of the box" and could be used to validate a new installation. Such an example sends a persistent JMS message to a queue called `queue/testQueue`.

To run the example and validate the installation, open an new command line window and set the `JBOSS_HOME` environment variable to point to the JBoss AS 4.x installation you've just installed Messaging on. Navigate to the folder where you extracted the release bundle and drill down to `/examples/queue`. Apache Ant must pe present in your path in order to be able to run the example.

Make sure you start the JBoss server before trying to run the tests

```
setenv JBOSS_HOME=<your_JBoss_installation>
cd .../examples/queue
$ant
```

A successfull execution log output looks similar to:

```
[tim@Vigor14 queue]$ ant
Buildfile: build.xml

identify:
     [echo] ############################################################################
     [echo] #                       Running the QUEUE example                          #
     [echo] ############################################################################
     [echo] The queue:      testQueue
     [echo] The client jar: ../../../output/lib/jboss-messaging-client.jar

sanity-check:

init:
    [mkdir] Created dir: /home/tim/dev/jboss-messaging/trunk/docs/examples/queue/output/classes
    [mkdir] Created dir: /home/tim/dev/jboss-messaging/trunk/docs/examples/common/output/classes

compile:
    [javac] Compiling 5 source files to /home/tim/dev/jboss-messaging/trunk/docs/examples/common/output/c
    [javac] Compiling 1 source file to /home/tim/dev/jboss-messaging/trunk/docs/examples/queue/output/cla

run:
     [java] Queue /queue/testQueue exists
     [java] The message was successfully sent to the testQueue queue
     [java] Received message: Hello!
     [java] The example connected to JBoss Messaging version 1.3.0.GA (1.3)

     [java] #####################
     [java] ###    SUCCESS!   ###
     [java] #####################

BUILD SUCCESSFUL
Total time: 5 seconds
[tim@Vigor14 queue]$
```

It is recommended to run `all` validation examples available in the `example` directory (`queue`, `topic`, `mdb`, `stateless`, etc.). In Chapter 6, we will have a look at each of those examples.

5.5. Accessing JBoss Messaging from a remote client

In order to access JBoss Messaging from a client outside the JBoss app server, you will need to ensure the following jar files are on the client classpath:

- jboss-messaging-client.jar - This is available in the messaging distribution
- jbossall-client.jar - This is available in your $JBOSS_HOME/client directory
- $JBOSS_HOME/server/<SERVER_NAME>/deploy/jboss-aop.deployer/jboss-aop.jar

  JBoss AOP 1.5.5.GA+

  http://repository.jboss.com/jboss/aop/1.5.5.GA/lib/

  (For AOP, sometimes you have to use a specific JAR according to your JVM of choice. Use the most convenient for you)
- $JBOSS_HOME/server/<SERVER_NAME>/lib/javassist.jar

  Javassist 3.5.0.GA-brew+

  http://repository.jboss.com/javassist/3.5.0.GA-brew/lib/
- $JBOSS_HOME/server/<SERVER_NAME>/lib/trove.jar

  trove 1.0.2-brew

  http://repository.jboss.com/trove/1.0.2-brew/lib/

- log4j

6

Running the Examples

In the directory `docs/examples`, you will find a set of examples demonstrating JBoss Messaging working in various examples, they include:

- docs/example/queue

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

- docs/example/topic

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

- docs/example/mdb

  This example demonstrates usage of an EJB2.1 MDB with JBoss Messaging

- docs/example/ejb3mdb

  This example demonstrates usage of an EJB3 MDB with JBoss Messaging

- docs/example/stateless

  This example demonstrates an EJB2.1 stateless session bean interacting with JBoss Messaging

- docs/example/mdb-failure

  This example demonstrates rollback and redelivery occuring with an EJB2.1 MDB

- docs/example/secure-socket

  This example demonstrates a JMS client interacting with a JBoss Messaging server using SSL encrypted transport

- docs/example/http

  This example demonstrates a JMS client interacting with a JBoss Messaging server tunneling traffic over the HTTP protocol

- docs/example/web-service

  This example demonstrates JBoss web-service interacting with JBoss Messaging

- docs/example/distributed-queue

  This example demonstrates a JMS client interacting with a JBoss Messaging distributed queue - it requires two

JBoss AS instances to be running

- docs/example/distributed-topic

  This example demonstrates a JMS client interacting with a JBoss Messaging distributed topic - it requires two JBoss AS instances to be running

- docs/example/stateless-clustered

  This example demonstrates a JMS client interacting with clustered EJB2.1 stateless session bean, which in turn interactes with JBoss Messaging. The example uses HAJNDI to lookup the connection factory

- docs/example/bridge

  This example demonstrates using a message bridge. It deploys a message bridge in JBoss AS which then proceeds to move messages from a source to a target queue

It is highly recommended that you familiarise yourself with the examples.

Make sure you start the JBoss server(s) before running the examples!

The non clustered examples expect a JBoss AS instance to be running with all the default settings

The clustered examples expect two JBoss AS instances to running with ports settings as per ports-01 and ports-02.

For each example, you can always override the default ports it will try to connect to by editing jndi.properties in the particular example directory

7

Configuration

The JMS API specifies how a messaging client interacts with a messaging server. The exact definition and implementation of messaging services, such as message destinations and connection factories, are specific to JMS providers. JBoss Messaging has its own configuration files to configure services. If you are migrating services from JBossMQ (or other JMS provider) to JBoss Messaging, you will need to understand those configuration files.

In this chapter, we discuss how to configure various services inside JBoss Messaging, which work together to provide JMS API level services to client applications.

The JBoss Messaging service configuration is spread among several configuration files. Depending on the functionality provided by the services it configures, the configuration data is distributed between `messaging-service.xml`, `remoting-service.xml`, `xxx-persistence-service.xml` (or `clustered-xxx-persistence-service.xml` for a clustered configuration, more about clusterered configuration in Chapter 8) (where xxx is the name of the database you are using) `connection-factories-service.xml` and `destinations-service.xml`.

The AOP client-side and server-side interceptor stacks are configured in `aop-messaging-client.xml` and `aop-messaging-server.xml`. Normally you will not want to change them, but you can some of the interceptors can be removed to give a small performance increase, if you don't need them. Be very careful you have considered the security implications before removing the security interceptor.

7.1. Configuring the ServerPeer

The Server Peer is the "heart" of the JBoss Messaging JMS facade. The server's configuration, resides in `messaging-service.xml` configuration file.

All JBoss Messaging services are rooted at the server peer

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

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

   <constructor>
      <!-- ServerPeerID -->
      <arg type="int" value="0"/>
      <!-- DefaultQueueJNDIContext -->
      <arg type="java.lang.String" value="/queue"/>
      <!-- DefaultTopicJNDIContext -->
      <arg type="java.lang.String" value="/topic"/>
   </constructor>
```

```
       <attribute name="PostOffice">jboss.messaging:service=PostOffice</attribute>
   <attribute name="SecurityDomain">java:/jaas/messaging</attribute>
   <attribute name="DefaultSecurityConfig">
     <security>
         <role name="guest" read="true" write="true" create="true"/>
     </security>
   </attribute>
   <attribute name="DefaultDLQ">jboss.messaging.destination:service=Queue,name=DLQ</attribute>
   <attribute name="DefaultMaxDeliveryAttempts">10</attribute>
   <attribute name="DefaultExpiryQueue">jboss.messaging.destination:service=Queue,name=ExpiryQueue</at
   <attribute name="DefaultRedeliveryDelay">0</attribute>
   <attribute name="QueueStatsSamplePeriod">5000</attribute>
   <attribute name="FailoverStartTimeout">60000</attribute>
   <attribute name="FailoverCompleteTimeout">300000</attribute>
   <attribute name="DefaultMessageCounterHistoryDayLimit">-1</attribute>

   <depends optional-attribute-name="PersistenceManager">jboss.messaging:service=PersistenceManager</d
   <depends optional-attribute-name="JMSUserManager">jboss.messaging:service=JMSUserManager</depends>
   <depends>jboss.messaging:service=Connector,transport=bisocket</depends>

</mbean>
```

7.1.1. ServerPeer attributes and arguments

We now discuss the MBean attributes and constructor arguments of the ServerPeer MBean

7.1.1.1. PersistenceManager

This is the persistence manager that the ServerPeer uses. You will not normally need to change this attribute.

7.1.1.2. PostOffice

This is the post office that the ServerPeer uses. You will not normally need to change this attribute. The post office is responsible for routing messages to queues and maintaining the mapping between addresses and queues.

7.1.1.3. JMSUserManager

This is the JMS user manager that the ServerPeer uses. You will not normally need to change this attribute.

7.1.1.4. DefaultDLQ

This is the name of the default DLQ (Dead Letter Queue) the server peer will use for destinations. The DLQ can be overridden on a per destination basis - see the destination MBean configuration for more details. A DLQ is a special destination where messages are sent when the server has attempted to deliver them unsuccessfully more than a certain number of times. If the DLQ is not specified at all then the message will be removed after the maximum number of delivery attempts. The maximum number of delivery attempts can be specified using the attribute DefaultMaxDeliveryAttempts for a global default or individually on a per destination basis.

7.1.1.5. DefaultExpiryQueue

This is the name of the default expiry queue the server peer will use for destinations. The expiry can be overridden on a per destination basis - see the destination MBean configuration for more details. An expiry queue is a special destination where messages are sent when they have expired. Message expiry is determined by the value of Message::getJMSExpiration() If the expiry queue is not specified at all then the message will be removed after it is expired.

7.1.1.6. ServerPeerID

The unique id of the server. In a cluster each server MUST have a unique id.

7.1.1.7. DefaultQueueJNDIContext

The default JNDI context to use when binding queues. Defaults to /queue.

7.1.1.8. DefaultTopicJNDIContext

The default JNDI context to use when binding topics. Defaults to /topic.

7.1.1.9. Destinations

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

7.1.1.10. DefaultMaxDeliveryAttempts

The default for the maximum number of times delivery of a message will be attempted before sending the message to the DLQ, if configured.

The default value is `10`

This value can also be overridden on a per destination basis

7.1.1.11. FailoverStartTimeout

The maximum number of milliseconds the client will wait for failover to start on the server side when a problem is detected.

The default value is `60000` (one minute)

7.1.1.12. FailoverCompleteTimeout

The maximum number of milliseconds the client will wait for failover to complete on the server side after it has started.

The default value is `300000` (five minutes)

7.1.1.13. DefaultRedeliveryDelay

When redelivering a message after failure of previous delivery it is often beneficial to introduce a delay perform redelivery in order to prevent thrashing of delivery-failure, delivery-failure etc

The default value is `0` which means there will be no delay.

Change this if your application could benefit with a delay before redelivery. This value can also be overridden on a per destination basis

7.1.1.14. QueueStatsSamplePeriod

Periodically the server will query each queue to gets its message statistics. This is the period.

The default value is `10000` milliseconds

7.1.1.15. DefaultMessageCounterHistoryDayLimit

JBoss Messaging provides a message counter history which shows the number of messages arriving on each queue of a certain number of days. This attribute represents the maxiumum number of days for which to store message counter history. It can be overridden on a per destination basis

7.1.1.16. MessageCounters

JBoss Messaging provides a message counter for each queue.

7.1.1.17. MessageCountersStatistics

JBoss Messaging provides statistics for each message counter for each queue.

7.1.1.18. DefaultSecurityConfig

Default security configuration is used when the security configuration for a specific queue or topic has not been overridden in the destination's deployment descriptor. It has exactly the same syntax and semantics as in JBossMQ.

The `DefaultSecurityConfig` attribute element should contain one `<security>` element. The `<security>` element can contain multiple `<role>` elements. Each `<role>` element defines the default access for that particular role.

If the `read` attribute is `true` then that role will be able to read (create consumers, receive messaages or browse) destinations by default.

If the `write` attribute is `true` then that role will be able to write (create producers or send messages) to destinations by default.

If the `create` attribute is `true` then that role will be able to create durable subscriptions on topics by default.

7.1.2. We now discuss the MBean operations of the ServerPeer MBean

7.1.2.1. DeployQueue

This operation lets you programmatically deploy a queue.

There are two overloaded versions of this operation

If the queue already exists but is undeployed it is deployed. Otherwise it is created and deployed

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

The `jndiName` parameter (optional) represents the full jndi name where to bind the destination. If this is not specified then the destination will be bound in <DefaultQueueJNDIContext>/<name>

The first version of this operation deploys the destination with the default paging parameters. The second overloaded version deploys the destination with the specified paging parameters. See the section on configuring destinations for a discussion of what the paging parameters mean

7.1.2.2. UndeployQueue

This operation lets you programmatically undeploy a queue.

The queue is undeployed but is NOT removed from persistent storage.

This operation returns `true` if the queue was successfull undeployed. otherwise it returns `false`

7.1.2.3. DestroyQueue

This operation lets you programmatically destroy a queue.

The queue is undeployed and then all its data is destroyed from the database

> **Warning**
>
> Be careful when using this method since it will delete all data for the queue

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

7.1.2.4. DeployTopic

This operation lets you programmatically deploy a topic.

There are two overloaded versions of this operation

If the topic already exists but is undeployed it is deployed. Otherwise it is created and deployed

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

The `jndiName` parameter (optional) represents the full jndi name where to bind the destination. If this is not specified then the destination will be bound in <DefaultTopicJNDIContext>/<name>

The first version of this operation deploys the destination with the default paging parameters. The second overloaded version deploys the destination with the specified paging parameters. See the section on configuring destinations for a discussion of what the paging parameters mean

7.1.2.5. UndeployTopic

This operation lets you programmatically undeploy a topic.

The queue is undeployed but is NOT removed from persistent storage.

This operation returns `true` if the topic was successfully undeployed. otherwise it returns `false`

7.1.2.6. DestroyTopic

This operation lets you programmatically destroy a topic.

The topic is undeployed and then all its data is destroyed from the database

Warning

Be careful when using this method since it will delete all data for the topic

This operation returns `true` if the topic was successfully destroyed. otherwise it returns `false`

7.1.2.7. ListMessageCountersHTML

This operation returns message counters in an easy to display HTML format.

7.1.2.8. ResetAllMesageCounters

This operation resets all message counters to zero.

7.1.2.9. ResetAllMesageCounters

This operation resets all message counter histories to zero.

7.1.2.10. EnableMessageCounters

This operation enables all message counters for all destinations. Message counters are disabled by default.

7.1.2.11. DisableMessageCounters

This operation disables all message counters for all destinations. Message counters are disabled by default.

7.1.2.12. RetrievePreparedTransactions

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

7.1.2.13. ShowPreparedTransactions

Retrieves a list of the Xids for all transactions currently in a prepared state on the node in an easy to display HTML format.

7.2. Changing the Database

Several JBoss Messaging services interact with persistent storage. They include: The Persistence Manager, The PostOffice and the JMS User Manager. The Persistence Manager is used to handle the message-related persistence. The Post Office handles binding related persistence. The JMS User manager handles user related persistence The configuration for all these MBeans is handled in the `xxx-persistence-service.xml` file

If the database you want to switch to is one of MySQL, Oracle, PostgreSQL, MS SQL Sever or Sybase, persistence configuration files are already available in the `examples/config` directory of the release bundle.

In order to enable support for one of these databases, just replace the default `hsqldb-persistence-service.xml` configuration file with the database-specific configuration file and restart the server.

Also, be aware that by default, the Messaging services relying on a datastore are referencing `"java:/DefaultDS"` for the datasource. If you are deploying a datasource with a different JNDI name, you need to update all the `DataSource` attribute in the persistence configuration file. Example data source configurations for each of the popular databases are available in the distribution.

7.3. Configuring the Post office

It is the job of the post office to route messages to their destination(s).

The post office maintains the mappings between addresses to which messages can be sent and their final queues.

For example when sending a message with an address that represents a JMS queue name, the post office will route this to a single queue - the JMS queue. When sending a message with an address that repesents a JMS topic name, the post office will route this to a set of queues - one for each JMS subscription.

The post office also handles the persistence for the mapping of addresses

JBoss Messaging comes with two different post office implementations - depending on whether you want clustering or not. The clustered post office has the ability to route messages to other nodes in the cluster

Whether you use the clustered post office or not depends on whether you deploy the `clustered-xxx-persistence-service.xml` MBean config or just the non clustered `xxx-persistence-service.xml` file.

It is likely that in future releases of JBoss Messaging the clustered and non clustered post offices will be combined into a single post office for ease of configuration

7.3.1. Non clustered post office

The non clustered post office should be used where clustering is not required. It will be used when the non clustered `xxx-persistence-service.xml` file is deployed.

Here is an example of a non clustered post office configuration:

```

   <mbean code="org.jboss.messaging.core.plugin.DefaultPostOfficeService"
      name="jboss.messaging:service=PostOffice"
      xmbean-dd="xmdesc/DefaultPostOffice-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>
      <attribute name="PostOfficeName">JMS</attribute>
      <attribute name="DataSource">java:/DefaultDS</attribute>
      <attribute name="CreateTablesOnStartup">true</attribute>
      <attribute name="SqlProperties"><![CDATA[
CREATE_POSTOFFICE_TABLE=CREATE TABLE JBM_POSTOFFICE (POSTOFFICE_NAME VARCHAR(255), NODE_ID INTEGER, QUEUE
INSERT_BINDING=INSERT INTO JBM_POSTOFFICE (POSTOFFICE_NAME, NODE_ID, QUEUE_NAME, COND, SELECTOR, CHANNEL_
DELETE_BINDING=DELETE FROM JBM_POSTOFFICE WHERE POSTOFFICE_NAME=? AND NODE_ID=? AND QUEUE_NAME=?
LOAD_BINDINGS=SELECT NODE_ID, QUEUE_NAME, COND, SELECTOR, CHANNEL_ID, CLUSTERED FROM JBM_POSTOFFICE WHERE
```

```
      ]]></attribute>
   </mbean>
```

7.3.1.1. The non clustered post office has the following attributes

7.3.1.1.1. DataSource

The datasource the postoffice should use for persisting its mapping data

7.3.1.1.2. SQLProperties

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

7.3.1.1.3. CreateTablesOnStartup

Set this to `true` if you wish the post office to attempt to create the tables (and indexes) when it starts. If the tables (or indexes) already exist a `SQLException` will be thrown by the JDBC driver and ignored by the Persistence Manager, allowing it to continue.

By default the value of `CreateTablesOnStartup` attribute is set to `true`

7.3.1.1.4. PostOfficeName

The name of the post office

7.3.2. Clustered post office

The clustered post office should be used where clustering is required. It will be used when the clustered `clustered-xxx-persistence-service.xml` file is deployed.

Here is an example of a clustered post office configuration:

```
   <mbean code="org.jboss.messaging.core.plugin.ClusteredPostOfficeService"
      name="jboss.messaging:service=PostOffice"
      xmbean-dd="xmdesc/ClusteredPostOffice-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>
      <attribute name="PostOfficeName">Clustered JMS</attribute>
      <attribute name="DataSource">java:/DefaultDS</attribute>
      <attribute name="CreateTablesOnStartup">true</attribute>
      <attribute name="SqlProperties"><![CDATA[
CREATE_POSTOFFICE_TABLE=CREATE TABLE JBM_POSTOFFICE (POSTOFFICE_NAME VARCHAR(255), NODE_ID INTEGER, QUEUE
INSERT_BINDING=INSERT INTO JBM_POSTOFFICE (POSTOFFICE_NAME, NODE_ID, QUEUE_NAME, COND, SELECTOR, CHANNEL_
DELETE_BINDING=DELETE FROM JBM_POSTOFFICE WHERE POSTOFFICE_NAME=? AND NODE_ID=? AND QUEUE_NAME=?
LOAD_BINDINGS=SELECT NODE_ID, QUEUE_NAME, COND, SELECTOR, CHANNEL_ID, CLUSTERED FROM JBM_POSTOFFICE WHERE
      ]]></attribute>
      <attribute name="GroupName">DefaultPostOffice</attribute>
      <attribute name="StateTimeout">5000</attribute>
      <attribute name="CastTimeout">5000</attribute>
      <attribute name="StatsSendPeriod">3000</attribute>
      <attribute name="MessagePullPolicy">org.jboss.messaging.core.plugin.postoffice.cluster.NullMessageP
      <attribute name="ClusterRouterFactory">org.jboss.messaging.core.plugin.postoffice.cluster.DefaultRo


      <attribute name="ChannelFactoryName">jgroups.mux:name=Multiplexer</attribute>
      <attribute name="SyncChannelName">udp-sync</attribute>
      <attribute name="AsyncChannelName">udp</attribute>
      <attribute name="ChannelPartitionName">${jboss.partition.name:DefaultPartition}-JMS</attribute>

      <attribute name="AsyncChannelConfig">
         <config>
            <UDP mcast_recv_buf_size="500000" down_thread="false" ip_mcast="true" mcast_send_buf_size="32
           mcast_port="45567" ucast_recv_buf_size="500000" use_incoming_packet_handler="false"
           mcast_addr="228.8.8.8" use_outgoing_packet_handler="true" loopback="true" ucast_send_buf_size=
            <AUTOCONF down_thread="false" up_thread="false"/>
            <PING timeout="2000" down_thread="false" num_initial_members="3" up_thread="false"/>
            <MERGE2 max_interval="10000" down_thread="false" min_interval="5000" up_thread="false"/>
            <FD_SOCK down_thread="false" up_thread="false"/>
            <FD timeout="20000" max_tries="3" down_thread="false" up_thread="false" shun="true"/>
            <VERIFY_SUSPECT timeout="1500" down_thread="false" up_thread="false"/>
            <pbcast.NAKACK max_xmit_size="8192" down_thread="false" use_mcast_xmit="true" gc_lag="50" up_
                         retransmit_timeout="100,200,600,1200,2400,4800"/>
            <UNICAST timeout="1200,2400,3600" down_thread="false" up_thread="false"/>
            <pbcast.STABLE stability_delay="1000" desired_avg_gossip="20000" down_thread="false" max_byte
            <FRAG frag_size="8192" down_thread="false" up_thread="false"/>
            <VIEW_SYNC avg_send_interval="60000" down_thread="false" up_thread="false" />
            <pbcast.GMS print_local_addr="true" join_timeout="3000" down_thread="false" join_retry_timeou
         </config>
      </attribute>

      <attribute name="SyncChannelConfig">
         <config>
            <UDP mcast_recv_buf_size="500000" down_thread="false" ip_mcast="true" mcast_send_buf_size="32
           mcast_port="45568" ucast_recv_buf_size="500000" use_incoming_packet_handler="false"
           mcast_addr="228.8.8.8" use_outgoing_packet_handler="true" loopback="true" ucast_send_buf_size=
            <AUTOCONF down_thread="false" up_thread="false"/>
            <PING timeout="2000" down_thread="false" num_initial_members="3" up_thread="false"/>
            <MERGE2 max_interval="10000" down_thread="false" min_interval="5000" up_thread="false"/>
            <FD_SOCK down_thread="false" up_thread="false"/>
            <FD timeout="20000" max_tries="3" down_thread="false" up_thread="false" shun="true"/>
            <VERIFY_SUSPECT timeout="1500" down_thread="false" up_thread="false"/>
            <pbcast.NAKACK max_xmit_size="8192" down_thread="false" use_mcast_xmit="true" gc_lag="50" up_
                         retransmit_timeout="100,200,600,1200,2400,4800"/>
            <UNICAST timeout="1200,2400,3600" down_thread="false" up_thread="false"/>
            <pbcast.STABLE stability_delay="1000" desired_avg_gossip="20000" down_thread="false" max_byte
            <FRAG frag_size="8192" down_thread="false" up_thread="false"/>
            <VIEW_SYNC avg_send_interval="60000" down_thread="false" up_thread="false" />
            <pbcast.GMS print_local_addr="true" join_timeout="3000" down_thread="false" join_retry_timeou
            <pbcast.STATE_TRANSFER down_thread="false" up_thread="false"/>
         </config>
      </attribute>
   </mbean>
```

7.3.2.1. The clustered post office has the following attributes

7.3.2.1.1. DataSource

The datasource the postoffice should use for persisting its mapping data

7.3.2.1.2. SQLProperties

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

7.3.2.1.3. CreateTablesOnStartup

Set this to `true` if you wish the post office to attempt to create the tables (and indexes) when it starts. If the tables (or indexes) already exist a `SQLException` will be thrown by the JDBC driver and ignored by the Persistence Manager, allowing it to continue.

By default the value of `CreateTablesOnStartup` attribute is set to `true`

7.3.2.1.4. PostOfficeName

The name of the post office

7.3.2.1.5. NodeIDView

This returns set containing the node ids of all the nodes in the cluster

7.3.2.1.6. GroupName

All post offices in the cluster with the same group name will form a cluster together. Make sure the group name matches with all the nodes in the cluster you want to form a cluster with.

7.3.2.1.7. MessagePullPolicy

JBoss Messaging has the ability for queues on one node to pull messages from other nodes when they have exhausted their local messages.

This prevents messages from getting stranded on nodes with slow or no consumers, and balances out message processing across the cluster.

How, if and when messages are pulled from one node to another is determined by the MessagePullPolicy

By default this set to `org.jboss.messaging.core.plugin.postoffice.cluster.NullMessagePullPolicy` which is a dummy implementation which never pulls messages from one node to another.

Whether you need message redistribution or not depends on your application topology - please see the section on clustering configuration for more details

If you require message redistribution then set this value to `org.jboss.messaging.core.plugin.postoffice.cluster.NullMessagePullPolicy`

> **Warning**
>
> Enabling message redistribution can result in the strict JMS message ordering guarantee being lost (i.e. the order of receipt of messages from a particular producer is retained). If this is not acceptable then do not enable message redistribution.

7.3.2.1.8. ClusterRouterFactory

When a message arrives on a node - JBoss Messaging needs to decide whether to route it to a local queue or a remote queue on a different node.

This setting allows you to specify the factory that determines this routing

By default this set to `org.jboss.messaging.core.plugin.postoffice.cluster.DefaultRouterFactory` which always favours a local queue if one is available otherwise it round robins amongst other queues.

The particular router factory you require depends on your application topology - please see the section on clustering configuration for more details

Other values this attribute can be set to are `org.jboss.messaging.core.plugin.postoffice.cluster.RoundRobinRouterFactory` if you do not want to favour the local queue.

7.3.2.1.9. StateTimeout

The maximum time to wait when waiting for the group state to arrive when a node joins a pre-existing cluster.

The default value is `5000` milliseconds

7.3.2.1.10. CastTimeout

The maximum time to wait for a reply casting message synchronously

The default value is `5000` milliseconds

7.3.2.1.11. StatsSendPeriod

When clustering, each node in the cluster will broadcast statistics periodically to inform the other nodes of their queues and the number of messages in them. This data is then used by the message redistribution policy to redistribute messages if necessary. This value represents the number of milliseconds between statistics broadcasts.

The default value is `10000` milliseconds

7.3.2.1.12. SyncChannelConfig

JBoss Messaging uses JGroups for all group management. This contains the JGroups stack configuration for the synchronous channel.

The synchronous channel is used for sending request/reeciving responses from other nodes in the cluster

The details of the JGroups configuration won't be discussed here since it is standard JGroups configuration. Detailed information on JGroups can be found in JGroups release documentation or on-line at http://www.jgroups.org or http://wiki.jboss.org/wiki/Wiki.jsp?page=JGroups

7.3.2.1.13. AsyncChannelConfig

JBoss Messaging uses JGroups for all group management. This contains the JGroups stack configuration for the asynchronous channel.

The asynchronous channel is used for sending sending/receiving messages from other nodes in the cluster

The details of the JGroups configuration won't be discussed here since it is standard JGroups configuration. Detailed information on JGroups can be found in JGroups release documentation or on-line at http://www.jgroups.org or http://wiki.jboss.org/wiki/Wiki.jsp?page=JGroups

7.3.2.1.14. ThreadPoolSize

The post office uses a thread pool for dispatching requests/handling responses from the cluster. This attribute represents the maximum size of that pool

The default value of this is `50` threads

7.4. Configuring the Persistence Manager

It is the job of the persistence manager to manage all message related persistence.

JBoss Messaging ships with a JDBC Persistence Manager used for handling persistence of message data in a relational database accessed via JDBC. The Persistence Manager implementation is pluggable (the Persistence Manager is a Messaging server plug-in), this making possible to provide other implementations for persisting message data in non relational stores, file stores etc.

The configuration of "persistent" services is grouped in a `xxx-persistence-service.xml` file, where the actual file prefix is usually inferred from its corresponding database JDBC connection string. By default, Messaging ships with a `hsqldb-persistence-service.xml`, which configures the Messaging server to use the in-VM Hypersonic database instance that comes by default with any JBossAS instance.

> **Warning**
>
> The default Persistence Manager configuration is works out of the box with Hypersonic, however it must be stressed that Hypersonic should not be used in a production environment mainly due to its limited support for transaction isolation and its propensity to behave erratically under high load.
>
> The Critique of Hypersonic [http://wiki.jboss.org/wiki/Wiki.jsp?page=ConfigJBossMQDB] wiki page outlines some of the well-known issues occuring when using this database.

JBoss Messaging also ships with pre-made Persistence Manager configurations for MySQL, Oracle, PostgreSQL, Sybase and MS SQL Server. The example `mysql-persistence-service.xml`, `oracle-persistence-service.xml`, `postgres-persistence-service.xml` and `sybase-persistence-service.xml` and `mssql-persistence-service.xml` configuration files are available in the `examples/config` directory of the release bundle.

Users are encouraged to contribute their own configuration files where we will thoroughly test them before certifying them for suppported use with JBoss Messaging. The JDBC Persistence Manager has been designed to use standard SQL for the DML so writing a JDBC Persistence Manager configuration for another database is usually only a fairly simple matter of changing DDL in the configuration which is likely to be different for different databases.

The default Hypersonic persistence configuration file is listed below:

```

      <mbean code="org.jboss.messaging.core.plugin.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</dep
   <attribute name="DataSource">java:/DefaultDS</attribute>
   <attribute name="CreateTablesOnStartup">true</attribute>
   <attribute name="UsingBatchUpdates">false</attribute>
   <attribute name="MaxParams">500</attribute>
</mbean>
```

An example of a Persistence Manager configuration for a MySQL database follows:

```
<mbean code="org.jboss.messaging.core.plugin.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</
         <attribute name="DataSource">java:/DefaultDS</attribute>
         <attribute name="CreateTablesOnStartup">true</attribute>
         <attribute name="UsingBatchUpdates">true</attribute>
         <attribute name="SqlProperties"><![CDATA[
   CREATE_MESSAGE_REFERENCE=CREATE TABLE JBM_MSG_REF (CHANNEL_ID BIGINT, MESSAGE_ID BIGINT, TRANS
   CREATE_IDX_MESSAGE_REF_TX=CREATE INDEX JBM_MSG_REF_TX ON JBM_MSG_REF (TRANSACTION_ID)
   CREATE_IDX_MESSAGE_REF_ORD=CREATE INDEX JBM_MSG_REF_ORD ON JBM_MSG_REF (ORD)
   CREATE_IDX_MESSAGE_REF_PAGE_ORD=CREATE INDEX JBM_MSG_REF_PAGE_ORD ON JBM_MSG_REF (PAGE_ORD)
   CREATE_IDX_MESSAGE_REF_MESSAGE_ID=CREATE INDEX JBM_MSG_REF_MESSAGE_ID ON JBM_MSG_REF (MESSAGE_
   CREATE_IDX_MESSAGE_REF_SCHED_DELIVERY=CREATE INDEX JBM_MSG_REF_SCHED_DELIVERY ON JBM_MSG_REF (
   CREATE_MESSAGE=CREATE TABLE JBM_MSG (MESSAGE_ID BIGINT, RELIABLE CHAR(1), EXPIRATION BIGINT, T
   CREATE_TRANSACTION=CREATE TABLE JBM_TX (TRANSACTION_ID BIGINT, BRANCH_QUAL VARBINARY(254), FOR
   CREATE_COUNTER=CREATE TABLE JBM_COUNTER (NAME VARCHAR(255), NEXT_ID BIGINT, PRIMARY KEY(NAME))
   INSERT_MESSAGE_REF=INSERT INTO JBM_MSG_REF (CHANNEL_ID, MESSAGE_ID, TRANSACTION_ID, STATE, ORD
   DELETE_MESSAGE_REF=DELETE FROM JBM_MSG_REF WHERE MESSAGE_ID=? AND CHANNEL_ID=? AND STATE='C'
   UPDATE_MESSAGE_REF=UPDATE JBM_MSG_REF SET TRANSACTION_ID=?, STATE='-' WHERE MESSAGE_ID=? AND C
   UPDATE_PAGE_ORDER=UPDATE JBM_MSG_REF SET PAGE_ORD = ? WHERE MESSAGE_ID=? AND CHANNEL_ID=?
   COMMIT_MESSAGE_REF1=UPDATE JBM_MSG_REF SET STATE='C', TRANSACTION_ID = NULL WHERE TRANSACTION_
   COMMIT_MESSAGE_REF2=DELETE FROM JBM_MSG_REF WHERE TRANSACTION_ID=? AND STATE='-'
   ROLLBACK_MESSAGE_REF1=DELETE FROM JBM_MSG_REF WHERE TRANSACTION_ID=? AND STATE='+'
   ROLLBACK_MESSAGE_REF2=UPDATE JBM_MSG_REF SET STATE='C', TRANSACTION_ID = NULL WHERE TRANSACTIO
   LOAD_PAGED_REFS=SELECT MESSAGE_ID, DELIVERY_COUNT, PAGE_ORD, SCHED_DELIVERY FROM JBM_MSG_REF W
   LOAD_UNPAGED_REFS=SELECT MESSAGE_ID, DELIVERY_COUNT, SCHED_DELIVERY FROM JBM_MSG_REF WHERE STA
   LOAD_REFS=SELECT MESSAGE_ID, DELIVERY_COUNT, SCHED_DELIVERY FROM JBM_MSG_REF WHERE STATE = 'C'
   UPDATE_REFS_NOT_PAGED=UPDATE JBM_MSG_REF SET PAGE_ORD = NULL WHERE PAGE_ORD BETWEEN ? AND ? AN
   SELECT_MIN_MAX_PAGE_ORD=SELECT MIN(PAGE_ORD), MAX(PAGE_ORD) FROM JBM_MSG_REF WHERE CHANNEL_ID
   SELECT_EXISTS_REF=SELECT MESSAGE_ID FROM JBM_MSG_REF WHERE CHANNEL_ID = ? AND MESSAGE_ID = ?
   SELECT_EXISTS_REF_MESSAGE_ID=SELECT MESSAGE_ID FROM JBM_MSG_REF WHERE MESSAGE_ID = ?
   UPDATE_DELIVERY_COUNT=UPDATE JBM_MSG_REF SET DELIVERY_COUNT = ? WHERE CHANNEL_ID = ? AND MESSA
   UPDATE_CHANNEL_ID=UPDATE JBM_MSG_REF SET CHANNEL_ID = ? WHERE CHANNEL_ID = ?
   LOAD_MESSAGES=SELECT MESSAGE_ID, RELIABLE, EXPIRATION, TIMESTAMP, PRIORITY, HEADERS, PAYLOAD,
   INSERT_MESSAGE=INSERT INTO JBM_MSG (MESSAGE_ID, RELIABLE, EXPIRATION, TIMESTAMP, PRIORITY, HEA
   INC_CHANNEL_COUNT=UPDATE JBM_MSG SET CHANNEL_COUNT = CHANNEL_COUNT + 1 WHERE MESSAGE_ID=?
   DEC_CHANNEL_COUNT=UPDATE JBM_MSG SET CHANNEL_COUNT = CHANNEL_COUNT - 1 WHERE MESSAGE_ID=?
   DELETE_MESSAGE=DELETE FROM JBM_MSG WHERE MESSAGE_ID=? AND CHANNEL_COUNT=0
   MESSAGE_ID_COLUMN=MESSAGE_ID
   MESSAGE_EXISTS=SELECT MESSAGE_ID FROM JBM_MSG WHERE MESSAGE_ID = ? FOR UPDATE
   INSERT_TRANSACTION=INSERT INTO JBM_TX (TRANSACTION_ID, BRANCH_QUAL, FORMAT_ID, GLOBAL_TXID) VA
   DELETE_TRANSACTION=DELETE FROM JBM_TX WHERE TRANSACTION_ID = ?
   SELECT_PREPARED_TRANSACTIONS=SELECT TRANSACTION_ID, BRANCH_QUAL, FORMAT_ID, GLOBAL_TXID FROM J
   SELECT_MESSAGE_ID_FOR_REF=SELECT MESSAGE_ID, CHANNEL_ID FROM JBM_MSG_REF WHERE TRANSACTION_ID
   SELECT_MESSAGE_ID_FOR_ACK=SELECT MESSAGE_ID, CHANNEL_ID FROM JBM_MSG_REF WHERE TRANSACTION_ID
   UPDATE_COUNTER=UPDATE JBM_COUNTER SET NEXT_ID = ? WHERE NAME=?
   SELECT_COUNTER=SELECT NEXT_ID FROM JBM_COUNTER WHERE NAME=? FOR UPDATE
```

```
            INSERT_COUNTER=INSERT INTO JBM_COUNTER (NAME, NEXT_ID) VALUES (?, ?)
            SELECT_ALL_CHANNELS=SELECT DISTINCT(CHANNEL_ID) FROM JBM_MSG_REF
                  ]]></attribute>
                  <attribute name="MaxParams">500</attribute>
            </mbean>
```

7.4.1. We now discuss the MBean attributes of the PersistenceManager MBean

7.4.1.1. CreateTablesOnStartup

Set this to `true` if you wish the Persistence Manager to attempt to create the tables (and indexes) when it starts. If the tables (or indexes) already exist a `SQLException` will be thrown by the JDBC driver and ignored by the Persistence Manager, allowing it to continue.

By default the value of `CreateTablesOnStartup` attribute is set to `true`

7.4.1.2. UsingBatchUpdates

Set this to `true` if the database supports JDBC batch updates. The JDBC Persistence Manager will then group multiple database updates in batches to aid performance.

By default the value of `UsingBatchUpdates` attribute is set to `false`

7.4.1.3. UsingBinaryStream

Set this to `true` if you want messages to be store and read using a JDBC binary stream rather than using getBytes(), setBytes(). Some database has limits on the maximum number of bytes that can be get/set using getBytes()/setBytes().

By default the value of `UsingBinaryStream` attribute is set to `true`

7.4.1.4. UsingTrailingByte

Certain version of Sybase are known to truncate blobs if they have trailing zeros. To prevent this if this attribute is set to `true` then a trailing non zero byte will be added and removed to each blob before and after persistence to prevent the database from truncating it. Currently this is only known to be necessary for Sybase.

By default the value of `UsingTrailingByte` attribute is set to `false`

7.4.1.5. SQLProperties

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

7.4.1.6. MaxParams

When loading messages the persistence manager will generate prepared statements with many parameters. This value tells the persistence manager what the absolute maximum number of parameters are allowable per prepared statement.

By default the value of `MaxParams` attribute is set to `100`

7.5. Configuring the JMS user manager

The JMS user manager handles the mapping of pre-configured client IDs to users and also managers the user and role tables which may or may not be used depending on which login module you have configured

Here 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"><![CDATA[
CREATE_USER_TABLE=CREATE TABLE JBM_USER (USER_ID VARCHAR(32) NOT NULL, PASSWD VARCHAR(32) NOT NULL, CLIEN
CREATE_ROLE_TABLE=CREATE TABLE JBM_ROLE (ROLE_ID VARCHAR(32) NOT NULL, USER_ID VARCHAR(32) NOT NULL, PRIM
SELECT_PRECONF_CLIENTID=SELECT CLIENTID FROM JBM_USER WHERE USER_ID=?
POPULATE.TABLES.1=INSERT INTO JBM_USER (USER_ID,PASSWD,CLIENTID) VALUES ('dilbert','dogbert','dilbert-id'
      ]]></attribute>
   </mbean>
```

7.5.1. We now discuss the MBean attributes of the JMSUserManager MBean

7.5.1.1. CreateTablesOnStartup

Set this to `true` if you wish the JMS user manager to attempt to create the tables (and indexes) when it starts. If the tables (or indexes) already exist a `SQLException` will be thrown by the JDBC driver and ignored by the Persistence Manager, allowing it to continue.

By default the value of `CreateTablesOnStartup` attribute is set to `true`

7.5.1.2. UsingBatchUpdates

Set this to `true` if the database supports JDBC batch updates. The JDBC Persistence Manager will then group multiple database updates in batches to aid performance.

By default the value of `UsingBatchUpdates` attribute is set to `false`

7.5.1.3. SQLProperties

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

Default user and role data can also be specified here. Any data to be inserted must be specified with property names starting with `POPULATE.TABLES` as in the above example.

7.6. Configuring Destinations

7.6.1. Pre-configured destinations

JBoss Messaging ships with a default set of pre-configured destinations that will be deployed during the server start up. The file that contains configuration for these destinations is `destinations-service.xml`. A section of this file is listed below:

```
<!--
   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>


<mbean code="org.jboss.jms.server.destination.TopicService"
   name="jboss.messaging.destination:service=Topic,name=testTopic"
   xmbean-dd="xmdesc/Topic-xmbean.xml">
   <depends optional-attribute-name="ServerPeer">jboss.messaging:service=ServerPeer</depends>
   <depends>jboss.messaging:service=PostOffice</depends>
   <attribute name="SecurityConfig">
      <security>
         <role name="guest" read="true" write="true"/>
         <role name="publisher" read="true" write="true" create="false"/>
         <role name="durpublisher" read="true" write="true" create="true"/>
      </security>
   </attribute>
</mbean>

<mbean code="org.jboss.jms.server.destination.TopicService"
   name="jboss.messaging.destination:service=Topic,name=securedTopic"
   xmbean-dd="xmdesc/Topic-xmbean.xml">
   <depends optional-attribute-name="ServerPeer">jboss.messaging:service=ServerPeer</depends>
   <depends>jboss.messaging:service=PostOffice</depends>
   <attribute name="SecurityConfig">
      <security>
         <role name="publisher" read="true" write="true" create="false"/>
      </security>
   </attribute>
</mbean>


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

```
            <role name="guest" read="true" write="true"/>
            <role name="publisher" read="true" write="true" create="false"/>
            <role name="noacc" read="false" write="false" create="false"/>
         </security>
      </attribute>
   </mbean>

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


   <!-- It's possible for indiviual queues and topics to use a specific queue for
   an expiry or DLQ -->

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

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

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

   <mbean code="org.jboss.jms.server.destination.TopicService"
      name="jboss.messaging.destination:service=Topic,name=TopicWithOwnDLQAndExpiryQueue"
      xmbean-dd="xmdesc/Topic-xmbean.xml">
      <depends optional-attribute-name="ServerPeer">jboss.messaging:service=ServerPeer</depends>
      <depends>jboss.messaging:service=PostOffice</depends>
      <attribute name="DLQ">jboss.messaging.destination:service=Queue,name=PrivateDLQ</attribute>
      <attribute name="ExpiryQueue">jboss.messaging.destination:service=Queue,name=PrivateExpiryQueue</at
   </mbean>


   <mbean code="org.jboss.jms.server.destination.TopicService"
      name="jboss.messaging.destination:service=Topic,name=TopicWithOwnRedeliveryDelay"
      xmbean-dd="xmdesc/Topic-xmbean.xml">
      <depends optional-attribute-name="ServerPeer">jboss.messaging:service=ServerPeer</depends>
      <depends>jboss.messaging:service=PostOffice</depends>
      <attribute name="RedeliveryDelay">5000</attribute>
   </mbean>


   <mbean code="org.jboss.jms.server.destination.TopicService"
      name="jboss.messaging.destination:service=Topic,name=testDistributedTopic"
      xmbean-dd="xmdesc/Topic-xmbean.xml">
      <depends optional-attribute-name="ServerPeer">jboss.messaging:service=ServerPeer</depends>
      <depends>jboss.messaging:service=PostOffice</depends>
      <attribute name="Clustered">true</attribute>
   </mbean>
....
```

7.6.2. Configuring queues

7.6.2.1. We now discuss the attributes of the Queue MBean

7.6.2.1.1. Name

The name of the queue

7.6.2.1.2. JNDIName

The JNDI name where the queue is bound

7.6.2.1.3. DLQ

The DLQ used for this queue. Overrides any value set on the ServerPeer config

7.6.2.1.4. ExpiryQueue

The Expiry queue used for this queue. Overrides any value set on the ServerPeer config

7.6.2.1.5. RedeliveryDelay

The redlivery delay to be used for this queue. Overrides any value set on the ServerPeer config

7.6.2.1.6. Destination Security Configuration

`SecurityConfig` - allows you to determine which roles are allowed to read, write and create on the destination. It has exactly the same syntax and semantics as the security configuration in JBossMQ destinations.

The `SecurityConfig` element should contain one `<security>` element. The `<security>` element can contain multiple `<role>` elements. Each `<role>` element defines the access for that particular role.

If the `read` attribute is `true` then that role will be able to read (create consumers, receive messaages or browse) this destination.

If the `write` attribute is `true` then that role will be able to write (create producers or send messages) to this destination.

If the `create` attribute is `true` then that role will be able to create durable subscriptions on this destination.

Note that the security configuration for a destination is optional. If a `SecurityConfig` element is not specifed then the default security configuration from the Server Peer will be used.

7.6.2.1.7. Destination paging parameters

'Pageable Channels' are a sophisticated new feature available in JBoss Messaging.

If your application needs to support very large queues or subscriptions containing potentially millions of messages, then it's not possible to store them all in memory at once.

JBoss Messaging solves this problem but letting you specify the maximum number of messages that can be stored in memory at any one time, on a queue-by-queue, or topic-by-topic basis. JBoss Messaging then pages messages to and from storage transparently in blocks, allowing queues and subscriptions to grow to very large sizes without any performance degradation as channel size increases.

This has been tested with in excess of 10 million 2K messages on very basic hardware and has the potential to scale to much larger number of messages.

The individual parameters are:

`FullSize` - this is the maximum number of messages held by the queue or topic subscriptions in memory at any one time. The actual queue or subscription can hold many more messages than this but these are paged to and from storage as necessary as messages are added or consumed.

`PageSize` - When loading messages from the queue or subscrition this is the maximum number of messages to pre-load in one operation.

`DownCacheSize` - When paging messages to storage from the queue they first go into a "Down Cache" before being written to storage. This enables the write to occur as a single operation thus aiding performance. This setting determines the max number of messages that the Down Cache will hold before they are flushed to storage.

If no values for `FullSize`, `PageSize`, or `DownCacheSize` are specified they will default to values 75000, 2000, 2000 respectively.

If you want to specify the paging parameters used for temporary queues then you need to specify them on the appropriate connection factory. See connection factory configuration for details.

7.6.2.1.8. CreatedProgrammatically

Returns `true` if the queue was created programmatically

7.6.2.1.9. MessageCount

Returns the total number of messages in the queue = number not being delivered + number being delivered + number being scheduled

7.6.2.1.10. 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 is a feature of JBoss Messaging where you can send a message and specify the earliest time at which it will be delivered. E.g. you can send a message now, but the message won't actually be delivered until 2 hours time.

To do this, you just need to set the following header in the message before sending:

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

                Message msg = sess.createMessage();

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

                prod.send(msg);
```

7.6.2.1.11. MaxSize

A maximum size (in number of messages) can be specified for a queue. Any messages that arrive beyond this point will be dropped. The default is `-1` which is unbounded.

7.6.2.1.12. Clustered

Clustered destinations must have this set to `true`

7.6.2.1.13. MessageCounter

Each queue maintains a message counter.

7.6.2.1.14. MessageCounterStatistics

The statistics for the message counter

7.6.2.1.15. MessageCounterHistoryDayLimit

The maximum number of days to hold message counter history for. Overrides any value set on the ServerPeer.

7.6.2.1.16. ConsumerCount

The number of consumers currently consuming from the queue.

7.6.2.2. We now discuss the MBean operations of the Queue MBean

7.6.2.2.1. RemoveAllMessages

Remove (and delete) all messages from the queue.

Warning

Use this with caution. It will permanently delete all messages from the queue

.

7.6.2.2.2. ListAllMessages

List all messages currently in the queue

There are two overloaded versions of this operation: One takes a JMS selector as an argument, the other does not. By using the selector you can retrieve a subset of the messages in the queue that match the criteria

7.6.2.2.3. ListDurableMessages

As listAllMessages but only lists the durable messages

There are two overloaded versions of this operation: One takes a JMS selector as an argument, the other does not. By using the selector you can retrieve a subset of the messages in the queue that match the criteria

7.6.2.2.4. ListNonDurableMessages

As listAllMessages but only lists the non durable messages

There are two overloaded versions of this operation: One takes a JMS selector as an argument, the other does not. By using the selector you can retrieve a subset of the messages in the queue that match the criteria

7.6.2.2.5. ResetMessageCounter

Resets the message counter to zero.

7.6.2.2.6. ResetMessageCounterHistory

Resets the message counter history.

7.6.2.2.7. ListMessageCounterAsHTML

Lists the message counter in an easy to display HTML format

7.6.2.2.8. ListMessageCounterHistoryAsHTML

Lists the message counter history in an easy to display HTML format

7.6.3. Configuring topics

7.6.3.1. We now discuss the MBean attributes of the Topic MBean

7.6.3.1.1. Name

The name of the topic

7.6.3.1.2. JNDIName

The JNDI name where the topic is bound

7.6.3.1.3. DLQ

The DLQ used for this topic. Overrides any value set on the ServerPeer config

7.6.3.1.4. ExpiryQueue

The Expiry queue used for this topic. Overrides any value set on the ServerPeer config

7.6.3.1.5. RedeliveryDelay

The redelivery delay to be used for this topic. Overrides any value set on the ServerPeer config

7.6.3.1.6. Destination Security Configuration

`SecurityConfig` - allows you to determine which roles are allowed to read, write and create on the destination. It has exactly the same syntax and semantics as the security configuration in JBossMQ destinations.

The `SecurityConfig` element should contain one `<security>` element. The `<security>` element can contain multiple `<role>` elements. Each `<role>` element defines the access for that particular role.

If the `read` attribute is `true` then that role will be able to read (create consumers, receive messaages or browse) this destination.

If the `write` attribute is `true` then that role will be able to write (create producers or send messages) to this destination.

If the `create` attribute is `true` then that role will be able to create durable subscriptions on this destination.

Note that the security configuration for a destination is optional. If a `SecurityConfig` element is not specifed then the default security configuration from the Server Peer will be used.

7.6.3.1.7. Destination paging parameters

'Pageable Channels' are a sophisticated new feature available in JBoss Messaging.

If your application needs to support very large queues or subscriptions containing potentially millions of messages, then it's not possible to store them all in memory at once.

JBoss Messaging solves this problem but letting you specify the maximum number of messages that can be stored in memory at any one time, on a queue-by-queue, or topic-by-topic basis. JBoss Messaging then pages messages to and from storage transparently in blocks, allowing queues and subscriptions to grow to very large sizes without any performance degradation as channel size increases.

This has been tested with in excess of 10 million 2K messages on very basic hardware and has the potential to scale to much larger number of messages.

The individual parameters are:

`FullSize` - this is the maximum number of messages held by the queue or topic subscriptions in memory at any one time. The actual queue or subscription can hold many more messages than this but these are paged to and from storage as necessary as messages are added or consumed.

`PageSize` - When loading messages from the queue or subscrition this is the maximum number of messages to preload in one operation.

`DownCacheSize` - When paging messages to storage from the queue they first go into a "Down Cache" before being written to storage. This enables the write to occur as a single operation thus aiding performance. This setting determines the max number of messages that the Down Cache will hold before they are flushed to storage.

If no values for `FullSize`, `PageSize`, or `DownCacheSize` are specified they will default to values 75000, 2000, 2000 respectively.

If you want to specify the paging parameters used for temporary queues then you need to specify them on the appropriate connection factory. See connection factory configuration for details.

7.6.3.1.8. CreatedProgrammatically

Returns `true` if the topic was created programmatically

7.6.3.1.9. MaxSize

A maximum size (in number of messages) can be specified for a topic subscription. Any messages that arrive beyond this point will be dropped. The default is `-1` which is unbounded.

7.6.3.1.10. Clustered

Clustered destinations must have this set to `true`

7.6.3.1.11. MessageCounterHistoryDayLimit

The maximum number of days to hold message counter history for. Overrides any value set on the ServerPeer.

7.6.3.1.12. MessageCounters

Return a list of the message counters for the subscriptions of this topic.

7.6.3.1.13. AllMessageCount

Return the total number of messages in all subscriptions of this topic.

7.6.3.1.14. DurableMessageCount

Return the total number of durable messages in all subscriptions of this topic.

7.6.3.1.15. NonDurableMessageCount

Return the total number of non durable messages in all subscriptions of this topic.

7.6.3.1.16. AllSubscriptionsCount

The count of all subscriptions on this topic

7.6.3.1.17. DurableSubscriptionsCount

The count of all durable subscriptions on this topic

7.6.3.1.18. NonDurableSubscriptionsCount

The count of all non durable subscriptions on this topic

7.6.3.2. We now discuss the MBean operations of the Topic MBean

7.6.3.2.1. RemoveAllMessages

Remove (and delete) all messages from the subscriptions of this topic.

> **Warning**
>
> Use this with caution. It will permanently delete all messages from the topic

7.6.3.2.2. ListAllSubscriptions

List all subscriptions of this topic

7.6.3.2.3. ListDurableSubscriptions

List all durable subscriptions of this topic

7.6.3.2.4. ListNonDurableSubscriptions

List all non durable subscriptions of this topic

7.6.3.2.5. ListAllSubscriptionsAsHTML

List all subscriptions of this topic in an easy to display HTML format

7.6.3.2.6. ListDurableSubscriptionsAsHTML

List all durable subscriptions of this topic in an easy to display HTML format

7.6.3.2.7. ListNonDurableSubscriptionsAsHTML

List all non durable subscriptions of this topic in an easy to display HTML format

7.6.3.2.8. ListAllMessages

Lists all messages for the specified subscription.

There are two overloaded versions of this operation. One that takes a selector and one that does not. By specifyingthe selector you can limit the messages returned.

7.6.3.2.9. ListNonDurableMessages

Lists all non durable messages for the specified subscription.

There are two overloaded versions of this operation. One that takes a selector and one that does not. By specifyingthe selector you can limit the messages returned.

7.6.3.2.10. ListDurableMessages

Lists all durable messages for the specified subscription.

There are two overloaded versions of this operation. One that takes a selector and one that does not. By specifyingthe selector you can limit the messages returned.

7.6.4. Configuring Connection Factories

With the default configuration JBoss Messaging binds two connection factories in JNDI at start-up.

The first connection factory is the default non-clustered connection factory and is bound into the following JNDI contexts: `/ConnectionFactory`, `/XAConnectionFactory`, `java:/ConnectionFactory`, `java:/XAConnectionFactory`. This connection factory is provided to maintain compatibility with applications originally written against JBoss MQ which has no automatic failover or load balancing. This connection factory should be used if you do not require client side automatic failover or load balancing.

The second connection factory is the default clustered connection factory and is bound into the following JNDI contexts `/ClusteredConnectionFactory`, `/ClusteredXAConnectionFactory`, `java:/ClusteredConnectionFactory`, `java:/ClusteredXAConnectionFactory`.

You may want to configure additional connection factories, for instance if you want to provide a default client id for a connection factory, or if you want to bind it in different places in JNDI, if you want different connection factories to use different transports, or if you want to selective enable or disable load-balancing and/or automatic failover for a particular connection factory. Deploying a new connection factory is equivalent with adding a new ConnectionFactory MBean configuration to `connection-factories-service.xml`.

It is also possible to create an entirely new service deployment descriptor `xxx-service.xml` altogether and deploy it in `$JBOSS_HOME/server/messaging/deploy`.

Connection factories can support automatic failover and/or load-balancing by setting the corresponding attributes

An example connection factory configuration is presented below:

```

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

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

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

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

```
   </mbean>
```

The above example would create a connection factory with pre-configured client ID `myClientID` and bind the connection factory in two places in the JNDI tree: `/MyConnectionFactory` and `/factories/cf`. The connection factory overrides the default values for PreFetchSize, DefaultTempQueueFullSize, DefaultTempQueuePageSize, DefaultTempQueueDownCacheSize and DupsOKBatchSize, SupportsFailover, SupportsLoadBalancing and LoadBalancingFactory. The connection factory will use 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.

7.6.4.1. We now discuss the MBean attributes of the ConnectionFactory MBean

7.6.4.1.1. ClientID

Connection factories can be pre-configured with a client id. Any connections created using this connection factory will obtain this client id

7.6.4.1.2. JNDIBindings

The list of the JNDI bindings for this connection factory

7.6.4.1.3. PrefetchSize

Each client side consumer maintains a local buffer of messages from which it consumes. The server typically sends messages as fast as it can to the consumer, and when the consumer is full it sends the server a "stop" message to say it is full. When it clears enough space it sends a "start" message to ask the server to continue sending messages. The prefetchSize determines the size of this buffer. Larger values give better throughput.

7.6.4.1.4. Temporary queue paging parameters

DefaultTempQueueFullSize, DefaultTempQueuePageSize, DefaultTempQueueDownCacheSize are optional attributes that determine the default paging parameters to be used for any temporary destinations scoped to connections created using this connection factory. See the section on paging channels for more information on what these values mean. They will default to values of 200000, 2000 and 2000 respectively if ommitted.

7.6.4.1.5. DupsOKBatchSize

When using a session with acknowledge mode of DUPS_OK_ACKNOWLEDGE this setting determines how many acknowledgments it will buffer locally before sending. The default value is `2000`

7.6.4.1.6. SupportsLoadBalancing

When using a connection factory with a clustered JBoss Messaging installation you can choose whether to enable client side connection load-balancing. This is determined by setting the attribute supportsLoadBalancing on the connection factory.

If load balancing is enabled on a connection factory then any connections created with that connection factory will be load-balanced across the nodes of the cluster. Once a connection is created on a particular node, it stays on that node.

The exact policy that determines how connections are load balanced is determined by the LoadBalancingFactory attribute

The default value is `false`

7.6.4.1.7. SupportsFailover

When using a connection factory with a clustered JBoss Messaging installation you can choose whether to enable client side automatic failover. This is determined by setting the attribute supportsFailover on the connection factory.

If automatic failover is enabled on a connection factory, then if a connection problem is detected with the connection then JBoss Messaging will automatically and transparently failover to another node in the cluster.

The failover is transparent meaning the user can carry on using the sessions, consumers, producers and connection objects as before.

If automatic failover is not required, then this attribute can be set to false. With automatic failover disabled it is up to the user code to catch connection exceptions in synchronous JMS operations and install a JMS ExceptionListener to catch exceptions asynchronously. When a connection is caught, the client side code should lookup a new connection factory using HAJNDI and recreate the connection using that.

The default value is `false`

7.6.4.1.8. LoadBalancingFactory

If you are using a connection factory with client side load balancing then you can specify how the load balancing is implemented by overriding this attribute. The value must correspond to the name of a class which implements the interface org.jboss.jms.client.plugin.LoadBalancingFactory

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

7.6.4.1.9. Connector

This specifies which remoting connector this connection factory uses. Different connection factories can use different connectors.

For instance you could deploy one connection factory that creates connections that use the HTTP transport to communicate to the server and another that creates connections that use the bisocket transport to communicate.

7.6.5. Configuring the remoting connector

JBoss Messaging uses JBoss Remoting for all client to server communication. For full details of what JBoss Remoting is capable of and how it is configured please consult the JBoss Remoting documentation.

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

The default connector is configured to use the remoting bisocket transport. The bisocket transport is a TCP socket based transport which only listens and accepts connections on the server side. I.e. connections are always initiated from the client side. This means it works well in typical firewall scenarios where only inbound connections are allowed on the server. Or where onlu outbound connections are allowed from the client.

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

The other supported transport is the HTTP transport. This uses the HTTP protocol to communicate between client and server. Data is received on the client by the client periodically polling the server for messages. This transport is well suited to situations where there is a firewall between client and server which only allows incoming HTTP traffic on the server. Please note this transport will not be as performant as the bisocket transport due to the nature of polling and the HTTP protocl. Also please note it is not designed for high load situations.

No other remoting transports are currently supported by JBoss Messaging

You can look at remoting configuration under:

<JBoss>/server/<YourMessagingServer>/deploy/jboss-messaging.sar/remoting-service.xml

By default JBoss Messaging binds to ${jboss.bind.address} which can be defined by: ./run.sh -c <yourconfig> -b yourIP.

You can change remoting-service.xml if you want for example use a different communication port.

> **Warning**
>
> Please be wary of changing other settings as they can have an adverse effect on the system

7.6.6. ServiceBindingManager

If you are using the JBoss AS ServiceBindingManager to provide different servers with different port ranges, then you must make sure that the JBoss Messaging remoting configuration specified in the JBoss Messaging section of the ServiceBindingManager xml file exactly matches that in remoting-service.xml

See the chapter on installation for a description of how to set-up the service binding manager for JBoss Messaging

7.6.7. Configuring the callback

JBoss Messaging uses a callback mechanism from Remoting that needs a Socket for callback operations. These socket properties are passed to the server by a remote call when the connection is being estabilished. As we said before we will support bidirectional protocols in future releases.

By default JBoss Messaging will execute InetAddress.getLocalHost().getHostAddress() to access your local host IP, but in case you need to setup a different IP, you can define a system property in your java arguments:

Use java -Djboss.messaging.callback.bind.address=YourHost - That will determine the callBack host in your client.

The client port will be selected randomly for any non used port. But if you defined -

Djboss.messaging.callback.bind.port=NumericPort in your System Properties that number is going to be used for the call back client port.

8

JBoss Messaging Clustering Configuration

In this chapter we will discuss how to configure JBoss Messaging clustering for different types of applications

Most of JBoss messaging clustering configuration revolves around the following variable:

- Choosing the connetion factory
- Choosing the cluster router policy
- Choosing the message redistributon policy

8.1. Choosing the connection factory

JBoss Messaging connections factories can optionally support load balancing and automatic failover. This is determined by the attributes on supportsLoadBalancing and supportsFailover on the connection factory.

See the section Section 7.6.4 for more information.

8.2. Choosing the cluster router policy

When a message is sent to a queue on particular node of the cluster, the system must decide whether the current node will handle it or whether it should send it to another node to handle.

The same applies if there are shared durable subscriptions on a topic and the message is being sent to a topic

The correct decision to make depends on your application topology.

If your application consists of a set of servers with the same MDBs deployed on each server, and you have many well distributed producers sending messages on all the nodes of the cluster, then the best policy is to always favour the local queue, since extra network trips are more expensive and the other nodes are also havng local messages sent to them

However if your application consists of a set of homogenous MDBs but you have few or badly distributed producers, then always favouring the local producer will mean the other nodes are being starved of messages and not using their CPUs cycles efficiently for messaging processing.

In this case, a good policy is to use a round robin routing policy where messages are round robin distributed to different nodes as they arrive.

In general, use the DefaultRoutingPolicy (this always favours the local queue) if you have many well distributed producers, and use the RoundRobinRoutingPolicy if you have few or badly distributed producers.

This are specified in the clustered post office config, by specifying the following attribute

To favour the local queue:

```
        <attribute name="ClusterRouterFactory">org.jboss.messaging.core.plugin.postoffice.cluster.DefaultR
```

To round robin:

```
          <attribute name="ClusterRouterFactory">org.jboss.messaging.core.plugin.postoffice.cluster.RoundR
```

8.3. Choosing the message pull policy

Once messages have arrived on queues in a cluster, in an ideal world they will all be consumed at the same rate, and there will be consumers on each node.

However, in some application topologies, consumes may close on queues on a node, leaving messages otherwise stranded, or perhaps consumers on some nodes are fast than consumers on other nodes causing messages to build up on one node or another and adversely effecting latency.

JBoss Messaging allows messages to pulled from one node to another as load dictates in order to cope with such situations

Whether or not you should activate message pulling (message redistribution) depends on your application topology

For an application that consists of a set of servers with a heterogenous bank of MDBs (or other consumers) deployed on each node, consuming at approximately the same rate, then message redistribution is not necessary.

However, if your application consists of different numbers or rates of consumers on different nodes then message redistribution may help

> **Warning**
>
> By its nature, message redistribution can result in messages being delivered in an order different to the strict ordering imposed by JMS. I.e. messages can, sometimes be delivered in an order different to that which they were produced by their producer. If this ordering is important to you then don't use message redistribution

In general, use message redistribution when your consumers are not well distributed across the cluster or if they have greatly varying rates.

Message redistribution is set by setting the following attribute in the clustered post office configuration:

For no message redistribution:

```
            <attribute name="MessagePullPolicy">org.jboss.messaging.core.plugin.postoffice.cluster.NullMes
```

For message redistribution:

```
            <attribute name="MessagePullPolicy">org.jboss.messaging.core.plugin.postoffice.cluster.Default
```

When selecting message redistribution you should also choose a value of `StatsSendPeriod` appropriately.

9

JBoss Messaging XA Recovery Configuration

This section describes how to configure JBoss Transactions in JBoss AS 4.2.0 to handle XA recovery for JBoss Messaging resources.

JBoss Transactions recovery manager can easily be configured to continually poll for and recover JBoss Messaging XA resources, this provides an extremely high level of durability of transactions.

Enabling JBoss Transactions Recovery Manager to recover JBoss Messaging resources is a very simple matter and involves adding a line to the file ${JBOSS_CONFIG}/conf/jbossjta-properties.xml

Here's an example section of a jbossjta-properties.xml file with the line added (note the whole file is not shown)

```
     <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.transa
        <!--
                        com.arjuna.ats.internal.jta.transaction.jts.TransactionManagerImple
                        -->
        <property name="com.arjuna.ats.jta.jtaUTImplementation" value="com.arjuna.ats.internal.jta.transa
        <!--
                        com.arjuna.ats.internal.jta.transaction.jts.UserTransactionImple
                        -->

        <!-- *** Add this line to enable recovery for JMS resources using DefaultJMSProvider *** -->
        <property name="com.arjuna.ats.jta.recovery.XAResourceRecovery.JBMESSAGING1" value="org.jboss.jms

    </properties>
```

In the above example the recovery manager will attempt to recover JMS resources using the JMSProviderLoader "DefaultJMSProvider"

DefaultJMSProvider is the default JMS provider loader that ships with JBoss AS and is defined in jms-ds.xml (or hajndi-jms-ds.xml in a clustered configuration). If you want to recovery using a different JMS provider loader - e.g. one corresponding to a remote JMS provider then just add another line and instead of DefaultJMSProvider specify the name of the remote JMS provider as specified in it's MBean configuration.

For each line you add, the name must be unique, so you could specify "com.arjuna.ats.jta.recovery.XAResourceRecovery.JBMESSAGING1", "com.arjuna.ats.jta.recovery.XAResourceRecovery.JBMESSAGING2, ..." etc.

In actual fact, the recovery also should work with any JMS provider that implements recoverable XAResources (i.e. it properly implements XAResource.recover()) , not just JBoss Messaging

10

JBoss Messaging Message Bridge Configuration

10.1. Message bridge overview

JBoss Messaging includes a fully functional message bridge

The function of the bridge is to consume messages from a source queue or topic, and send them to target queue or topic, typically on a different server.

The source and target servers do not have to be in the same cluster which makes bridging suitable for reliably sending messages from one cluster to another, for instance across a WAN.

A bridge is deployed inside a JBoss AS instance. The instance can be the same instance as either the source or target server. Or could be on a third, separate JBoss AS instance.

A bridge is deployed as an MBean inside JBoss AS. Deployment is trivial - just drop the MBean descriptor into the deploy directory of a JBoss configuration which contains JBoss Messaging.

An example in docs/example/bridge demonstrates a simple bridge being deployed in JBoss AS, and moving messages from the source to the target destination

The bridge can also be used to bridge messages from other non JBoss Messaging JMS servers, as long as they are JMS 1.1 compliant. This is as yet untested.

The bridge has built in resilience to failure so if the source or target server connetion is lost, e.g. due to network failure. Then the bridge can retry connecting to the source and/or target until they come back online. When this happens it will resume operation as normal.

The bridge can be configured with an optional selector, so it will only consume messages matching that selector

It can be configured to consume from a queue or a topic. When it consumes from a topic it can be configured to consume using a non durable or durable subscription

The bridge can be configured to bridge messages with one of three levels of quality of service, they are:

- QOS_AT_MOST_ONCE

  With this QoS mode messages will reach the destination from the source at most once. The messages are consumed from the source and acknowledged before sending to the destination. Therefore there is a possibility that if failure occurs between removing them from the source and them arriving at the destination they could be lost. Hence delivery will occur at most once. This mode is avilable for both persistent and non persistent messages.

- QOS_DUPLICATES_OK

  With this QoS mode, the messages are consumed from the source and then acknowledged after they have been successfully sent to the destination. Therefore there is a possibility that if failure occurs after sending to the destination but before acknowledging them, they could be sent again when the system recovers. I.e. the destination might receive duplicates after a failure. This mode is available for both persistent and non persistent messages.

- QOS_ONCE_AND_ONLY_ONCE

  This QoS mode ensures messages will reach the destination from the source once and only once. (Sometimes this mode is known as "exactly once"). If both the source and the destination are on the same JBoss Messaging server instance then this can be achieved by sending and acknowledging the messages in the same local transaction. If the source and destination are on different servers this is achieved by enlisting the sending and consuming sessions in a JTA transaction. The JTA transaction is controlled by JBoss Transactions JTA implementation which is a fully recovering transaction manager, thus providing a very high degree of durability. If JTA is required then both supplied connection factories need to be XAConnectionFactory implementations. This mode is only available for persistent messages. This is likely to be the slowest mode since it requires extra persistence for the transaction logging. If you require this level of QoS, please be sure to enable XA recovery with JBoss Transactions.

  Note

  For a specific application it may possible to provide once and only once semantics without using the QOS_ONCE_AND_ONLY_ONCE QoS level. This can be done by using the QOS_DUPLICATES_OK mode and then checking for duplicates at the destination and discarding them. Some JMS servers provide automatic duplicate message detection functionality, or this may be possible to implement on the application level by maintaining a cache of received message ids on disk and comparing received messages to them. The cache would only be valid for a certain period of time so this approach is not as watertight as using QOS_ONCE_AND_ONLY_ONCE but may be a good choice depending on your specific application.

10.2. Bridge deployment

A message bridge is easily deployed by dropping the MBean descriptor in the deploy directory of your JBoss AS installation which contains JBoss Messaging

10.3. Bridge configuration

In this section we describe how to configure the message bridge

Here is an example of a message bridge configuration, with all the attributes shown. Note that some are commented out for this configuration, since it is not appropriate to specify them all at once. Which ones are specified depends on the configuration you want.

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

     <!-- 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">cheesecake</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">hotdog</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
          -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 targ
     <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>

   </mbean>
```

We will now discuss each attribute

10.3.1. SourceProviderLoader

This is the object name of the JMSProviderLoader MBean that the bridge will use to lookup the source connection factory and source destination.

By default JBoss AS ships with one JMSProviderLoader, deployed in the file `jms-ds.xml` - this is the default local JMSProviderLoader. (This would be in `hajndi-jms-ds.xml` in a clustered configuration)

If your source destination is on different servers or even correspond to a different, non JBoss JMS provider, then you can deploy another JMSProviderLoader MBean instance which references the remote JMS provider, and reference that from this attribute. The bridge would then use that remote JMS provider to contact the source destination

Note that if you are using a remote non JBoss Messaging source or target and you wish once and only once delivery then that remote JMS provider must provide a fully functional JMS XA resource implementation that works remotely from the server - it is known that some non JBoss JMS providers do not provide such a resource

10.3.2. TargetProviderLoader

This is the object name of the JMSProviderLoader MBean that the bridge will use to lookup the target connection factory and target destination.

By default JBoss AS ships with one JMSProviderLoader, deployed in the file `jms-ds.xml` - this is the default local JMSProviderLoader. (This would be in `hajndi-jms-ds.xml` in a clustered configuration)

If your target destination is on a different server or even correspond to a different, non JBoss JMS provider, then you can deploy another JMSProviderLoader MBean instance which references the remote JMS provider, and reference that from this attribute. The bridge would then use that remote JMS provider to contact the target destination

Note that if you are using a remote non JBoss Messaging source or target and you wish once and only once delivery then that remote JMS provider must provide a fully functional JMS XA resource implementation that works remotely from the server - it is known that some non JBoss JMS providers do not provide such a resource

10.3.3. SourceDestinationLookup

This is the full JNDI lookup for the source destination using the SourceProviderLoader

An example would be /queue/mySourceQueue

10.3.4. TargetDestinationLookup

This is the full JNDI lookup for the target destination using the TargetProviderLoader

An example would be /topic/myTargetTopic

10.3.5. SourceUsername

This optional attribute is for when you need to specify the username for creating the source connection

10.3.6. SourcePassword

This optional attribute is for when you need to specify the password for creating the source connection

10.3.7. TargetUsername

This optional attribute is for when you need to specify the username for creating the target connection

10.3.8. TargetPassword

This optional attribute is for when you need to specify the password for creating the target connection

10.3.9. QualityOfServiceMode

This integer represents the desired quality of service mode

Possible values are QOS_AT_MOST_ONCE = 0, QOS_DUPLICATES_OK = 1, QOS_ONCE_AND_ONLY_ONCE = 2

Please see Section 10.1 for an explanation of what these mean.

10.3.10. Selector

This optional attribute can contain a JMS selector expression used for consuming messages from the source destination. Only messages that match the selector expression will be bridged from the source to the target destination

Please note it is always more performant to apply selectors on source topic subscriptions to source queue consumers.

The selector expression must follow the JMS selector syntax specified here: http://java.sun.com/j2ee/1.4/docs/api/javax/jms/Message.html

10.3.11. MaxBatchSize

This attribute specifies the maximum number of messages to consume from the source destination before sending them in a batch to the target destination. It's value must >= 1

10.3.12. MaxBatchTime

This attribute specifies the maximum number of milliseconds to wait before sending a batch to target, even if the number of messages consumed has not reached MaxBatchSize. It's value must can be -1 to represent 'wait forever', or >=1 to specify an actual time.

10.3.13. SubName

If the source destination represents a topic, and you want to consume from the topic using a durable subscription then this attribute represents the durable subscription name

10.3.14. ClientID

If the source destination represents a topic, and you want to consume from the topic using a durable subscription then this attribute represents the the JMS client ID to use when creating/looking up the durable subscription

10.3.15. FailureRetryInterval

This represents the amount of time in ms to wait between trying to recreate connections to the source or target servers when the bridge has detected they have failed

10.3.16. MaxRetries

This represents the number of times to attempt to recreate connections to the source or target servers when the bridge has detected they have failed. The bridge will give up after trying this number of times. -1 represents 'try forever'