JBoss Messaging 1.2 User's Guide

The next generation enterprise messaging solution from JBoss

Table of Contents

1

Introducing JBoss Messaging Release 1.2.0.GA

JBoss Messaging is a high performance JMS provider in the JBoss Enterprise Middleware Stack (JEMS). It is a complete rewrite of JBossMQ, which is the current default JMS provider in JBoss AS 4.0.x series and JBoss AS 4.2.0. JBoss Messaging will be the default JMS provider in JBoss AS 4.2.x series, probably starting with 4.2.1, and it is already the default provider in JBoss 5.0.0.Beta.

JBoss Messaging it is also the backbone of the JBoss ESB infrastructure.

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 Chapter 10 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.

While JBoss Messaging only becomes the default JMS provider from JBoss AS 4.2.1 and JBoss 5.0, production users on JBoss AS 4.0.x can still take advantage of the performance improvements by easily replacing the JBossMQ module with JBoss Messaging.

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 JBoss 4.0.x production 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].

Team:

Ovidiu Feodorov, Project Lead

Tim Fox, Technical Lead

Clebert Suconic, Core Messaging Developer

Other contributions by: Adrian Brock, Bela Ban, Alex Fu, Luc Texier, Aaron Walker, Rajdeep Dua, Madhusudhan Konda, Juha Lindfors and Ron Sigal.

This manual was written with help from Luc Texier, Lead Support EMEA and Sam Griffith Jr., JBoss Documentation Team.

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 will soon become an integral component of the JBoss Enterprise Middleware Suite (JEMS). Currently it is available for embedded use within the JBoss Application Server (JBossAS), and as a JBoss Microkernel-based stand-alone server. Work to integrate JBoss Messaging with the new JBoss Microcontainer is under way.

The large and vibrant JEMS developer community fosters its continued innovation and enterprise quality. JBoss Messaging enables more agile applications in a wide range of scenarios from simple messaging needs to an enterprise-wide messaging foundation and adds flexibility to any SOA initiative.

2.1. JBoss Messaging support cover from Red Hat

JBoss Messaging 1.2.GA will 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 be fully available for these plaforms and it will cover JBoss Messaging.

Red Hat is also working on introducing a development support program, which is different from production support and caters to companies who want to have a safety net even from the development phase. The estimated availability date of this support progam is April 2007.

2.2. JBoss Messaging 1.2.0.GA Features

JBoss Messaging provides:

- A fully compatible and Sun certified JMS 1.1 implementation, that currently works with a standard JBossAS 4.0.x, 4.2.x and JBoss 5.x installation and also as a JBoss Microkernel-based standalone deployment.
- A strong focus on performance, reliability and scalability with high throughput and low latency. JBoss Messaging already exceeds JBoss MQ in a number of measured performance metrics. Full results will follow.

- A foundation for JBoss ESB for SOA initiatives; JBoss ESB uses JBoss Messaging as its foundation.

JBoss Messaging consists of two major parts:

- JBoss Messaging Core – a transactional, reliable messaging transport system.
    - Supports generalized messages (not just JMS)
    - Enables other messaging protocol façades to be added
    - Distributed, transactional and reliable
- JMS Façade – the JMS "personality" of JBoss Messaging.

Other JBoss Messaging features include:

- Publish-subscribe and point-to-point messaging models
- Topics that feed multiple message queues
- Persistent and non-persistent messages
- Guaranteed message delivery that ensures that messages arrive once and only once
- 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

JBoss Messaging 1.2.0.GA Clustering provides the following 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 1.2 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

JBossMQ is the JMS implementation currently shipped within JBossAS. 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.

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 JBossMQ data schema and vice-versa.

2.4. Features not yet available in the 1.2.0.GA release, but coming soon

All the final features are available apart from:

- In a very near future, it will be possible to guarantee persistent level reliability guarantee without persistence. By replicating persistent messages between nodes in memory, we can obtain comparable reliability levels to persisting messages to disk, without actually storing them to disk. However, this feature is not available in the 1.2.0.GA release. If you're interested in how this feature will be available, this is the JIRA issue you can use to track it: http://jira.jboss.org/jira/browse/JBMESSAGING-574

- The "unreliable link scenario" http://jira.jboss.org/jira/browse/JBMESSAGING-676 whose development is still on-going on a parallel branch.

3

JBoss Messaging Clustering

This section of the userguide gives a brief overview of the features available in JBoss Messaging Clustering 1.2.0.GA. 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 1.2.

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. A client side load balancing policy will automatically chose a node to connect to (This is similar to how EJB clustering chooses a node).

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.

Normally, persistent messages are persisted in a shared database which is shared by all nodes in the cluster. JBoss Messaging 1.2.1 will contain an option where you can choose to not persist persistent messages in a database, but instead to replicate them between nodes of the cluster. The idea here is the network IO on a fast network should be much faster than persisting to disk. This solution should also be more scalable since different nodes replicate their messages onto different other nodes - there is no "master node". If the messages are replicated onto sufficient nodes and the hardware is set-up with UPS, then we believe a comparable reliability guarantee to persisting messages to disk can be achieved. Of course, this won't be suitable for all situations, but you use the best tool for the job.

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 address spaces 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 releases of JBoss Messaging are available as a free download from the JBoss Messaging project landing page [http://www.jboss.com/products/messaging].

The recommended download location is the JBoss Labs Messaging Project download zone: http://labs.jboss.com/portal/jbossmessaging/downloads We also maintain an alternate download location on sourceforge.net: http://sourceforge.net/project/showfiles.php?group_id=22866&package_id=157261

4.1. The JBoss Messaging Release Bundle

The JBoss Messaging release bundle (`jboss-messaging-1.2.x.zip`) will expand in a `jboss-messaging-1.2.x` directory that contains:

- `jboss-messaging-scoped.sar` - the scoped JBoss service archive that contains JBoss Messaging and its dependencies.

  ### 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.

  ### Note

  This is an important note on the Messaging client jar. JBoss Messaging has an extensive tree of dependencies and bundling all right dependencies in one easy to use archive, so no wrong versions inadvertently get in the class path is, as experience shows, extremely convenient. However, this could cause problems if a different version of a dependency is required by the user's client code.

  If, for example, the log4j.jar bundled with the release causes problems for being too old, one could just simply build the client-side class path so a newer version of log4j.jar precedes jboss-messaging-client.jar in that class path. This approach may work, but is not guaranteed to work, because we did not run the full test suite with any other log4j.jar version except the one we bundle with the release. Soon we will providing an ant task that creates a slimmed down client jar without the extra dependencies bundled.

  A list of required dependencies for a specific release can be found in the `build-thirdpary.xml` file that is part of the companion source bundle that ships with each release. It can also be looked up online in http://fisheye.jboss.org.

- `util` - a collection of `ant` configuration files used to automate installation and release management procedures. 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 relatively sophisticated use cases in which EJBs and JCA JMS ConnectionFactories are involved, as well as clustering use cases. The `examples/config` sub-directory contains various configuration file examples.
- `docs` - this user's guide.
- `src/jboss-messaging-1.2.x-src.zip` - the zipped source directory. The file can be directly installed into and used with a debugger.
- `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 Non-Clustered Installation

This section describes the procedure to perfom a non-clustered installation. At the end of this procedure, you will create a JBoss Messaging configuration that will start a non-clustered messaging server. To find out how you can install JBoss Messaging in a clustered configuration, please proceed to Chapter 6.

By default, a JBoss AS 4.0.x and 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.

Warning

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-scoped.sar` over to the JBoss instance `deploy` directory. Follow one of the alternate installation procedures outlined below instead.

5.1. Installing JBoss Messaging with JBoss AS 4.0.x and 4.2.1

Note

You need Apache ant 1.7.0 or newer pre-installed on your system and available in you path in order to be able to perform the installation process described below.

5.1.1. Installation procedure

Set up the `JBOSS_HOME` environment variable to point to the JBoss 4.x 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.

Note

If you want to create a JBoss Messaging configuration with a different name, modify the `messaging.config.name` system property declared at the beginning of the installation script accordingly.

5.2. 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 messaging
```

A successful JBoss Messaging deployment generates logging output similar to:

```
....
01:44:48,317 WARN  [JDBCPersistenceManager]

JBoss Messaging Warning: DataSource connection transaction isolation should be
       READ_COMMITTED, but it is currently NONE.
       Using an isolation level less strict than READ_COMMITTED may lead to data
       consistency problems.
       Using an isolation level more strict than READ_COMMITTED may lead to deadlock.

01:44:50,450 INFO  [ServerPeer] JBoss Messaging 1.2.0.GA server [0] started
01:44:51,311 INFO  [ConnectionFactory] Connector bisocket://192.168.1.104:4457 has
leasing enabled, lease period 10000 milliseconds
01:44:51,311 INFO  [ConnectionFactory] [/ConnectionFactory, /XAConnectionFactory,
java:/ConnectionFactory, java:/XAConnectionFactory] started
01:44:51,441 INFO  [ConnectionFactory] Connector bisocket://192.168.1.104:4457 has
leasing enabled, lease period 10000 milliseconds
01:44:51,441 INFO  [ConnectionFactory] [/NonClusteredConnectionFactory,
/NonClusteredXAConnectionFactory, java:/NonClusteredConnectionFactory,
java:/NonClusteredXAConnectionFactory] started
01:44:51,681 INFO  [QueueService] Queue[/queue/DLQ] started, fullSize=200000,
pageSize=2000, downCacheSize=2000
01:44:51,681 INFO  [QueueService] Queue[/queue/ExpiryQueue] started, fullSize=200000,
pageSize=2000, downCacheSize=2000
01:44:51,732 INFO  [TopicService] Topic[/topic/testTopic] started, fullSize=200000,
pageSize=2000, downCacheSize=2000
01:44:51,732 INFO  [TopicService] Topic[/topic/securedTopic] started, fullSize=200000,
pageSize=2000, downCacheSize=2000
01:44:51,732 INFO  [TopicService] Topic[/topic/testDurableTopic] started, fullSize=200000,
pageSize=2000, downCacheSize=2000
01:44:51,752 INFO  [QueueService] Queue[/queue/testQueue] started, fullSize=200000,
pageSize=2000, downCacheSize=2000
01:44:51,752 INFO  [QueueService] Queue[/queue/A] started, fullSize=200000,
pageSize=2000, downCacheSize=2000
01:44:51,762 INFO  [QueueService] Queue[/queue/B] started, fullSize=200000,
pageSize=2000, downCacheSize=2000
01:44:51,762 INFO  [QueueService] Queue[/queue/C] started, fullSize=200000,
pageSize=2000, downCacheSize=2000
01:44:51,882 INFO  [QueueService] Queue[/queue/D] started, fullSize=200000,
pageSize=2000, downCacheSize=2000
01:44:51,882 INFO  [QueueService] Queue[/queue/ex] started, fullSize=200000,
pageSize=2000, downCacheSize=2000
01:44:51,882 INFO  [QueueService] Queue[/queue/PrivateDLQ] started, fullSize=200000,
pageSize=2000, downCacheSize=2000
01:44:51,892 INFO  [QueueService] Queue[/queue/PrivateExpiryQueue] started, fullSize=200000,
pageSize=2000, downCacheSize=2000
01:44:51,892 INFO  [QueueService] Queue[/queue/QueueWithOwnDLQAndExpiryQueue] started,
fullSize=200000, pageSize=2000, downCacheSize=2000
01:44:51,892 INFO  [TopicService] Topic[/topic/TopicWithOwnDLQAndExpiryQueue] started,
fullSize=200000, pageSize=2000, downCacheSize=2000
01:44:51,942 INFO  [QueueService] Queue[/queue/QueueWithOwnRedeliveryDelay] started,
fullSize=200000, pageSize=2000, downCacheSize=2000
01:44:51,942 INFO  [TopicService] Topic[/topic/TopicWithOwnRedeliveryDelay] started,
fullSize=200000, pageSize=2000, downCacheSize=2000
```

```
01:44:51,952 INFO  [QueueService] Queue[/queue/testDistributedQueue] started, fullSize=200000,
pageSize=2000, downCacheSize=2000
01:44:51,952 INFO  [TopicService] Topic[/topic/testDistributedTopic] started, fullSize=200000,
pageSize=2000, downCacheSize=2000
01:44:52,372 INFO  [ConnectionFactoryBindingService] Bound ConnectionManager
'jboss.jca:name=JmsXA,service=ConnectionFactoryBinding' to JNDI name 'java:JmsXA'
....
```

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 8.2 for details about replacing Hypersonic.

5.3. 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.

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

A successfull execution log output looks similar to:

```
$ 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: c:\work\src\svn\messaging\docs\examples\queue\output
[mkdir] Created dir: c:\work\src\svn\messaging\docs\examples\common\output

compile:
[javac] Compiling 5 source files to c:\work\src\svn\messaging\docs\examples\common\output
[javac] Compiling 1 source file to c:\work\src\svn\messaging\docs\examples\queue\output

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 1.2.0.GA (1.2)
   [java]
   [java] ######################
   [java] ###    SUCCESS!   ###
   [java] ######################

BUILD SUCCESSFUL
Total time: 9 seconds
```

It is recommended to run `all` validation examples available in the `example` directory (`queue`, `topic`, `mdb`, `stateless`, etc.). In Chapter 7, we will have a look at each of those examples.

6

JBoss Messaging Clustered Installation

Note

You need Apache ant 1.7.0 or newer pre-installed on your system and available in you path in order to be able to perform the installation process described below.

Use the `release-admin.xml` ant script shipped with the release to create individual cluster node configurations.

The typical usage is:

```
cd util
ant -f release-admin.xml [-Did=node-id] [-Dports=port-config-label]
[-Ddatabase=db-name] cluster-node
```

where:

- `node-id` is the unique node ID, an integer that must be unique per cluster. If not specified, it defaults to 0.
- `port-config-label` is a binding manager server configuration label. The short story behind this parameter is the following: multiple application servers running on the same physical machine need to use different service port ranges to avoid port conflicts. You can configure the whole port range used by a server instance by enabling a special service, the binding management service, and specifiying a "server" configuration in the binding manager's configuration file, which will determine specific port values to use when starting that instance. The Messaging installation script can enable the service binding manager and performs all configuration changes automatically. You only need to specify the "server" configuration you want to use, as 'port-config-label'. If you plan to run your clustering nodes on different physical machines, this parameter is irrelevant, and you should not use it. However, if you install two (or more) nodes of your cluster on the same physical machine, you need to give the value corresponding to a specific "server" configurations in the binding manager configuration file. JBoss AS ships "out-of-the-box" with several pre-configured port ranges: 'ports-default', 'ports-01', 'ports-02', 'ports-03'. Use one of these. If -Dports is not specified, the clustered instance created this way will fall over to the default port range for a JBoss instace. More details about the binding management service can be found in the Application Server documentation, at the following address http://docs.jboss.com/jbossas/guides/j2eeguide/r2/en/html_single/#ch10.bindingmanager
- `db-name` is the name of the database you want to create the configurations for. Supported database names are: mysql, oracle, postgresql, mssql, and sybase. Notice that if you create database configuration to any database other than MySQL, you will still need to manually install the database JDBC drivers (MySQL drivers are provided as part of the distribution).

Notice that you should have environment variable JBOSS_HOME correctly set to point to your JBoss AS distribution directory before executing any of the scripts.

For example, in order to create the configuration for a four-node cluster intended to run on the same physical machine, use the following sequence:

```
ant -f release-admin.xml cluster-node
ant -f release-admin.xml -Did=1 -Dports=ports-01 cluster-node
ant -f release-admin.xml -Did=2 -Dports=ports-02 cluster-node
ant -f release-admin.xml -Did=3 -Dports=ports-03 cluster-node
```

The sequence will create four cluster node configurations ("messaging-node0", "messaging-node1", "messaging-node2" and "messaging-node3").

The first command will create a cluster node with ID equals to '0' and using the default JBoss AS port assignments.

Warning

The configuration that has just been created uses a generic mysql service descriptor. Check `$JBOSS_HOME/server/messaging-node<id>/deploy/mysql-ds.xml` and verify that that:

- 1. Your database is indeed mysql.
- 2. It is accessible from every physical node you installed Messaging on.
- 3. Contains a schema (database/tablespace) named 'messaging'.
- 4. The URL (hostname and port), username and password are correct.
- 5. The installed mysql-driver.jar's version maches your database.

To start the cluster, from four different terminals, run:

```
cd $JBOSS_HOME/bin
./run.sh -c messaging-node0

cd $JBOSS_HOME/bin
./run.sh -c messaging-node1

cd $JBOSS_HOME/bin
./run.sh -c messaging-node2

cd $JBOSS_HOME/bin
./run.sh -c messaging-node3
```

A successful two node cluster startup produces a log similar to:

Node 0:

```
...

00:24:04,796 WARN  [JDBCPersistenceManager]

JBoss Messaging Warning:
   DataSource connection transaction isolation should be READ_COMMITTED, but it
```

```
   is currently REPEATABLE_READ.
  Using an isolation level less strict than READ_COMMITTED may lead to data
   consistency problems.
  Using an isolation level more strict than READ_COMMITTED may lead to deadlock.

00:24:05,718 INFO  [ServerPeer] JBoss Messaging 1.2.0.CR1 server [0] started
00:24:06,328 INFO  [STDOUT]
-------------------------------------------------------
GMS: address is 127.0.0.1:2452
-------------------------------------------------------
00:24:08,406 INFO  [DefaultClusteredPostOffice] ClusteredPostOffice
[0:Clustered JMS:127.0.0.1:2452] got new view [127.0.0.1:2452|0] [127.0.0.1:2452]
00:24:08,468 INFO  [STDOUT]
-------------------------------------------------------
GMS: address is 127.0.0.1:2455
-------------------------------------------------------
00:24:10,906 INFO  [ConnectionFactory] Connector socket://10.11.14.105:4457 has leasing
enabled, lease period 10000 milliseconds
00:24:10,921 INFO  [ConnectionFactory] [/ConnectionFactory, /XAConnectionFactory,
java:/ConnectionFactory, java:/XAConnectionFactory] started
00:24:10,953 INFO  [QueueService] Queue[/queue/DLQ] started, fullSize=75000,
pageSize=2000, downCacheSize=2000
00:24:10,953 INFO  [QueueService] Queue[/queue/ExpiryQueue] started, fullSize=75000,
pageSize=2000, downCacheSize=2000
00:24:10,953 INFO  [TopicService] Topic[/topic/testTopic] started, fullSize=75000,
pageSize=2000, downCacheSize=2000
00:24:10,953 INFO  [TopicService] Topic[/topic/securedTopic] started, fullSize=75000,
pageSize=2000, downCacheSize=2000
00:24:10,968 INFO  [TopicService] Topic[/topic/testDurableTopic] started, fullSize=75000,
pageSize=2000, downCacheSize=2000
00:24:10,968 INFO  [QueueService] Queue[/queue/testQueue] started, fullSize=75000,
pageSize=2000, downCacheSize=2000
00:24:10,968 INFO  [QueueService] Queue[/queue/A] started, fullSize=75000,
pageSize=2000, downCacheSize=2000
00:24:10,968 INFO  [QueueService] Queue[/queue/B] started, fullSize=75000,
pageSize=2000, downCacheSize=2000
00:24:10,968 INFO  [QueueService] Queue[/queue/C] started, fullSize=75000,
pageSize=2000, downCacheSize=2000
00:24:10,968 INFO  [QueueService] Queue[/queue/D] started, fullSize=75000,
pageSize=2000, downCacheSize=2000
00:24:10,968 INFO  [QueueService] Queue[/queue/ex] started, fullSize=75000,
pageSize=2000, downCacheSize=2000
00:24:10,984 INFO  [QueueService] Queue[/queue/PrivateDLQ] started, fullSize=75000,
pageSize=2000, downCacheSize=2000
00:24:10,984 INFO  [QueueService] Queue[/queue/PrivateExpiryQueue] started,
fullSize=75000, pageSize=2000, downCacheSize=2000
00:24:10,984 INFO  [QueueService] Queue[/queue/QueueWithOwnDLQAndExpiryQueue] started,
fullSize=75000, pageSize=2000, downCacheSize=2000
00:24:10,984 INFO  [TopicService] Topic[/topic/TopicWithOwnDLQAndExpiryQueue] started,
fullSize=75000, pageSize=2000, downCacheSize=2000
00:24:10,984 INFO  [QueueService] Queue[/queue/QueueWithOwnRedeliveryDelay] started,
fullSize=75000, pageSize=2000, downCacheSize=2000
00:24:10,984 INFO  [TopicService] Topic[/topic/TopicWithOwnRedeliveryDelay] started,
fullSize=75000, pageSize=2000, downCacheSize=2000
00:24:11,000 INFO  [QueueService] Queue[/queue/testDistributedQueue] started, fullSize=75000,
pageSize=2000, downCacheSize=2000
00:24:11,000 INFO  [TopicService] Topic[/topic/testDistributedTopic] started, fullSize=75000,
pageSize=2000, downCacheSize=2000
00:24:11,093 INFO  [ConnectionFactoryBindingService] Bound
ConnectionManager'jboss.jca:name=JmsXA,service=ConnectionFactoryBinding' to JNDI
name 'java:JmsXA'
00:24:11,375 INFO  [TomcatDeployer] deploy, ctxPath=/jmx-console,
warUrl=.../deploy/jmx-console.war/
00:24:12,171 INFO  [Http11BaseProtocol] Starting Coyote HTTP/1.1 on http-0.0.0.0-8080
00:24:12,421 INFO  [ChannelSocket] JK: ajp13 listening on /0.0.0.0:8009
00:24:12,453 INFO  [JkMain] Jk running ID=0 time=0/47  config=null
```

```
00:24:12,515 INFO  [Server] JBoss (MX MicroKernel) [4.0.5.GA (build: CVSTag=Branch_4_0
date=200611221632)]
Started in 30s:375ms

00:27:21,343 INFO  [DefaultClusteredPostOffice] ClusteredPostOffice
[0:Clustered JMS:127.0.0.1:2452] got new view [127.0.0.1:2452|1]
[127.0.0.1:2452, 127.0.0.1:2474]
```

Node 1:

```
...

00:33:54,468 WARN  [JDBCPersistenceManager]

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

00:33:55,062 INFO  [ServerPeer] JBoss Messaging 1.2.0.CR1 server [1] started
00:33:55,609 INFO  [STDOUT]
-------------------------------------------------------
GMS: address is 127.0.0.1:2514
-------------------------------------------------------
00:33:57,734 INFO  [DefaultClusteredPostOffice]
ClusteredPostOffice[1:Clustered JMS:127.0.0.1:2514] got new
view [127.0.0.1:2452|3] [127.0.0.1:2452, 127.0.0.1:2514]
00:33:57,765 INFO  [STDOUT]
-------------------------------------------------------
GMS: address is 127.0.0.1:2519
-------------------------------------------------------
00:34:00,203 INFO  [ConnectionFactory] Connector socket://10.11.14.105:4557 has leasing
enabled, lease period 20000 milliseconds
00:34:00,203 INFO  [ConnectionFactory] [/ConnectionFactory, /XAConnectionFactory,
java:/ConnectionFactory, java:/XAConnectionFactory] started
00:34:00,234 INFO  [QueueService] Queue[/queue/DLQ] started, fullSize=75000, pageSize=2000,
downCacheSize=2000
00:34:00,234 INFO  [QueueService] Queue[/queue/ExpiryQueue] started, fullSize=75000,
pageSize=2000, downCacheSize=2000
00:34:00,234 INFO  [TopicService] Topic[/topic/testTopic] started, fullSize=75000,
pageSize=2000, downCacheSize=2000
00:34:00,250 INFO  [TopicService] Topic[/topic/securedTopic] started, fullSize=75000,
pageSize=2000, downCacheSize=2000
00:34:00,250 INFO  [TopicService] Topic[/topic/testDurableTopic] started, fullSize=75000,
pageSize=2000, downCacheSize=2000
00:34:00,250 INFO  [QueueService] Queue[/queue/testQueue] started, fullSize=75000,
pageSize=2000, downCacheSize=2000
00:34:00,250 INFO  [QueueService] Queue[/queue/A] started, fullSize=75000,
pageSize=2000, downCacheSize=2000
00:34:00,250 INFO  [QueueService] Queue[/queue/B] started, fullSize=75000,
pageSize=2000, downCacheSize=2000
00:34:00,250 INFO  [QueueService] Queue[/queue/C] started, fullSize=75000,
pageSize=2000, downCacheSize=2000
00:34:00,250 INFO  [QueueService] Queue[/queue/D] started, fullSize=75000,
pageSize=2000, downCacheSize=2000
00:34:00,250 INFO  [QueueService] Queue[/queue/ex] started, fullSize=75000,
pageSize=2000, downCacheSize=2000
00:34:00,265 INFO  [QueueService] Queue[/queue/PrivateDLQ] started, fullSize=75000,
pageSize=2000, downCacheSize=2000
00:34:00,265 INFO  [QueueService] Queue[/queue/PrivateExpiryQueue] started,
```

```
fullSize=75000, pageSize=2000, downCacheSize=2000
00:34:00,265 INFO  [QueueService] Queue[/queue/QueueWithOwnDLQAndExpiryQueue]
started, fullSize=75000, pageSize=2000, downCacheSize=2000
00:34:00,265 INFO  [TopicService] Topic[/topic/TopicWithOwnDLQAndExpiryQueue]
started, fullSize=75000, pageSize=2000, downCacheSize=2000
00:34:00,265 INFO  [QueueService] Queue[/queue/QueueWithOwnRedeliveryDelay]
started, fullSize=75000, pageSize=2000, downCacheSize=2000
00:34:00,265 INFO  [TopicService] Topic[/topic/TopicWithOwnRedeliveryDelay]
started, fullSize=75000, pageSize=2000, downCacheSize=2000
00:34:00,296 INFO  [QueueService] Queue[/queue/testDistributedQueue]
started, fullSize=75000, pageSize=2000, downCacheSize=2000
00:34:00,296 INFO  [TopicService] Topic[/topic/testDistributedTopic]
started, fullSize=75000, pageSize=2000, downCacheSize=2000
00:34:00,343 INFO  [ConnectionFactoryBindingService] Bound
ConnectionManager 'jboss.jca:name=JmsXA,
service=ConnectionFactoryBinding' to JNDI name 'java:JmsXA'
00:34:00,453 INFO  [TomcatDeployer] deploy, ctxPath=/jmx-console,
warUrl=.../deploy/jmx-console.war/
00:34:00,796 INFO  [Http11BaseProtocol] Starting Coyote HTTP/1.1 on http-0.0.0.0-8180
00:34:01,078 INFO  [ChannelSocket] JK: ajp13 listening on /0.0.0.0:8109
00:34:01,125 INFO  [JkMain] Jk running ID=0 time=0/125  config=null
00:34:01,125 INFO  [Server] JBoss (MX MicroKernel)
[4.0.5.GA (build: CVSTag=Branch_4_0 date=200611221632)] Started in 22s:547ms
```

Note

The installation script may fail while installing Messaging with source-generated JBoss 4.0.5.GA-ejb3 instance. This is because release-admin.xml relies on finding `$JBOSS_HOME/docs/examples/binding-manager/sample-bindings.xml`. 4.0.5.GA-ejb3 installations seem not to have a "docs" sub-directory. A very simple work-around for this situation is to recursively copy the "docs" sub-directory available under a regular (non-EJB3) source-generated 4.0.5.GA instance and retry the installation process.

7

Running the Examples

Since JBoss Messaging is a fully compliant JMS 1.1 provider, it supports the entire JMS 1.1 API. So, all JMS applications should work without modification. Integrated inside a JBoss AS, we should also be able access the JMS system from EJBs and write message-driven beans against JMS destinations.

In the following sections, we will look at examples of the various JMS messaging models and message-driven beans. They make use of pre-configured JMS destinations and connection factories that come default with the server. So, no extra configuration is needed to run those examples. Just set JBOSS_HOME and run `ant` in each example directory, as we described in Section 5.3. The example source directories are located in the distribution under `docs/examples`.

7.1. Sending messages to a queue

Open an new command line. Set the JBOSS_HOME environment variable to point at a JBossAS 4.x installation. Navigate to the folder where you exploded the main archive and drill down to `/examples/queue`. You need to use Apache `ant` to execute the `build.xml` file.

Make sure the JBoss server reference by the JBOSS_HOME is started.

```
public class QueueExample extends ExampleSupport
{

   public void example() throws Exception
   {
      String destinationName = getDestinationJNDIName();

      InitialContext ic = null;
      ConnectionFactory cf = null;
      Connection connection =  null;

      try
      {
         ic = new InitialContext();

         cf = (ConnectionFactory)ic.lookup("/ConnectionFactory");
         Queue queue = (Queue)ic.lookup(destinationName);
         log("Queue " + destinationName + " exists");

         connection = cf.createConnection();
         Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
         MessageProducer sender = session.createProducer(queue);

         TextMessage message = session.createTextMessage("Hello!");
         sender.send(message);
         log("The message was successfully sent to the " + queue.getQueueName() + " queue");

         MessageConsumer consumer =  session.createConsumer(queue);
```

```
         connection.start();

         message = (TextMessage)consumer.receive(2000);
         log("Received message: " + message.getText());
         assertEquals("Hello!", message.getText());

         displayProviderInfo(connection.getMetaData());

      }
      finally
      {
         if(ic != null)
         {
            try
            {
               ic.close();
            }
            catch(Exception e)
            {
               throw e;
            }
         }

         // ALWAYS close your connection in a finally block to avoid leaks.
         // Closing connection also takes care of closing its related objects e.g. sessions.
         closeConnection(connection);
      }
   }

   private void closeConnection(Connection con)
   {
      try
      {
         if (con != null)
         {
            con.close();
         }
      }
      catch(JMSException jmse)
      {
         log("Could not close connection " + con +" exception was " + jmse);
      }
   }

   protected boolean isQueueExample()
   {
      return true;
   }

   public static void main(String[] args)
   {
      new QueueExample().run();
   }

}
```

7.2. Sending messages to a topic

In this example, a standalone Java client publishes a text-based JMS message to a topic and a single subscriber pulls the message off the queue.

Open an new command line. Set the JBOSS_HOME environment variable to point at a JBossAS 4.x installation. Navigate to the folder where you exploded the main archive and drill down to `/examples/queue`. You need to use Apache Ant to execute the build.xml file Make sure the JBoss server reference by the JBOSS_HOME is started.

```
public class TopicExample extends ExampleSupport
{
   public void example() throws Exception
   {
      String destinationName = getDestinationJNDIName();

      InitialContext ic = null;
      Connection connection = null;

      try
      {
         ic = new InitialContext();

         ConnectionFactory cf = (ConnectionFactory)ic.lookup("/ConnectionFactory");
         Topic topic = (Topic)ic.lookup(destinationName);
         log("Topic " + destinationName + " exists");

         connection = cf.createConnection();
         Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
         MessageProducer publisher = session.createProducer(topic);
         MessageConsumer subscriber = session.createConsumer(topic);

         ExampleListener messageListener = new ExampleListener();
         subscriber.setMessageListener(messageListener);
         connection.start();

         TextMessage message = session.createTextMessage("Hello!");
         publisher.send(message);
         log("The message was successfully published on the topic");

         messageListener.waitForMessage();

         message = (TextMessage)messageListener.getMessage();
         log("Received message: " + message.getText());
         assertEquals("Hello!", message.getText());

         displayProviderInfo(connection.getMetaData());

      }
      finally
      {
         if (ic != null)
         {
            try
            {
               ic.close();
            }
            catch(Exception e)
            {
               throw e;
            }
         }

         // ALWAYS close your connection in a finally block to avoid leaks.
         // Closing connection also takes care of closing its related objects e.g. sessions.
         closeConnection(connection);
      }
   }

   private void closeConnection(Connection con) throws JMSException
```

```
   {

      try
      {
         if (con != null)
         {
            con.close();
         }
      }
      catch(JMSException jmse)
      {
         log("Could not close connection " + con +" exception was " + jmse);
      }
   }

   protected boolean isQueueExample()
   {
      return false;
   }

   public static void main(String[] args)
   {
      new TopicExample().run();
   }

}
```

7.3. Using JMS from an EJB

This example deploys a simple Stateless Session Bean that is used as a proxy to send and receive JMS messages in a managed environment.

Warning

Receiving a JMS message from inside an EJB (or MBean or servlet) method is considered an anti-pattern since it will require the creation of a new JMS consumer on every send - even when using the JCA adaptor. This is likely to make it very slow. (Users of the Spring JMSTemplate be warned that this is what it uses to implement it's receive.) If you want to receive messages as part of a global transaction then use an MDB instead.

```

public class StatelessSessionExampleBean implements SessionBean
{

   private ConnectionFactory cf = null;

   public void drain(String queueName) throws Exception
   {
      InitialContext ic = new InitialContext();
      Queue queue = (Queue)ic.lookup(queueName);
      ic.close();

      Session session = null;
      Connection conn = null;

      try
      {
         conn = getConnection();
         session = conn.createSession(false, Session.AUTO_ACKNOWLEDGE);
```

```
         MessageConsumer consumer = session.createConsumer(queue);
         Message m = null;
         do
         {
            m = consumer.receiveNoWait();
         }
         while(m != null);
      }
      finally
      {
         if (conn != null)
         {
            closeConnection(conn);
         }
      }
   }

   public void send(String txt, String queueName) throws Exception
   {
      InitialContext ic = new InitialContext();

      Queue queue = (Queue)ic.lookup(queueName);

      ic.close();

      Session session = null;
      Connection conn = null;

      try
      {
         conn = getConnection();

         session = conn.createSession(false, Session.AUTO_ACKNOWLEDGE);

         MessageProducer producer = session.createProducer(queue);

         TextMessage tm = session.createTextMessage(txt);

         producer.send(tm);

         System.out.println("message " + txt + " sent to " + queueName);
      }
      finally
      {
         if (conn != null)
         {
            closeConnection(conn);
         }
      }
   }

   public int browse(String queueName) throws Exception
   {
      InitialContext ic = new InitialContext();
      Queue queue = (Queue)ic.lookup(queueName);
      ic.close();

      Session session = null;
      Connection conn = null;

      try
      {
         conn = getConnection();
         session = conn.createSession(false, Session.AUTO_ACKNOWLEDGE);
         QueueBrowser browser = session.createBrowser(queue);
```

```
         ArrayList list = new ArrayList();
         for(Enumeration e = browser.getEnumeration(); e.hasMoreElements(); )
         {
            list.add(e.nextElement());
         }

         return list.size();
      }
      finally
      {
         if (conn != null)
         {
            closeConnection(conn);
         }
      }
   }

   public String receive(String queueName) throws Exception
   {
      InitialContext ic = new InitialContext();
      Queue queue = (Queue)ic.lookup(queueName);
      ic.close();

      Session session = null;
      Connection conn = null;

      // WARN! this is an anti-pattern - please see above warning

      try
      {
         conn = getConnection();
         session = conn.createSession(false, Session.AUTO_ACKNOWLEDGE);

         MessageConsumer consumer = session.createConsumer(queue);

         System.out.println("blocking to receive message from queue " + queueName + " ...");
         TextMessage tm = (TextMessage)consumer.receive(5000);

         if (tm == null)
         {
            throw new Exception("No message!");
         }

         System.out.println("Message " + tm.getText() + " received");

         return tm.getText();
      }
      finally
      {
         if (conn != null)
         {
            closeConnection(conn);
         }
      }
   }

   public Connection getConnection() throws Exception
   {

      Connection connection = null;

      try
      {
         connection = cf.createConnection();

         connection.start();
```

```
      }
      catch(Exception e )
      {
         if(connection != null)
         {
            closeConnection(connection);
         }
         System.out.println("Failed to get connection...exception is " + e);
         throw e;
      }

      return connection;
   }

   public void closeConnection(Connection con) throws Exception
   {
      try
      {
         con.close();
      }
      catch(JMSException jmse)
      {
         System.out.println("Could not close connection " + con +" exception was " + jmse);
         throw jmse;
      }
   }

   public void setSessionContext(SessionContext ctx) throws EJBException, RemoteException
   {
   }

   public void ejbCreate()
   {
      try
      {
         InitialContext ic = new InitialContext();

         cf = (ConnectionFactory)ic.lookup("java:/JmsXA");

         ic.close();
      }
      catch(Exception e)
      {
         e.printStackTrace();
         throw new EJBException("Initalization failure: " + e.getMessage());
      }
   }

   public void ejbRemove() throws EJBException
   {
      try
      {
         if(cf != null)
         {
            cf = null;
         }
      }
      catch(Exception e)
      {
         throw new EJBException("ejbRemove ", e);
      }
   }

   public void ejbActivate() throws EJBException, RemoteException
   {
   }
```

```
   public void ejbPassivate() throws EJBException, RemoteException
   {
   }

}
```

7.4. Using EJB2.1 Message Driven Beans

This example deploys a simple Message Driven Bean that processes messages sent to a test queue. Once it receives a message and "processes" it, the MDB sends an acknowledgment message to a temporary destination created by the sender for this purpose. The example is considered successful if the sender receives the acknowledgment message.

The MDB ejb-jar.xml descriptor

```
<ejb-jar>
  <enterprise-beans>
    <message-driven>
      <ejb-name>MDBExample</ejb-name>
      <ejb-class>org.jboss.example.jms.mdb.MDBExample</ejb-class>
      <transaction-type>Container</transaction-type>
    </message-driven>
  </enterprise-beans>
</ejb-jar>
```

The MDB jboss.xml descriptor

```
<enterprise-beans>
  <message-driven>
    <ejb-name>MDBExample</ejb-name>
    <destination-jndi-name>queue/@QUEUE_NAME@</destination-jndi-name>
  </message-driven>
</enterprise-beans>
```

```
public class MDBExample implements MessageDrivenBean, MessageListener
{
   private MessageDrivenContext ctx;

   private ConnectionFactory cf = null;

   public void onMessage(Message m)
   {
      Session session = null;
      Connection conn = null;

      try
      {
         TextMessage tm = (TextMessage)m;

         String text = tm.getText();
         System.out.println("message " + text + " received");
         String result = process(text);
```

```
         System.out.println("message processed, result: " + result);

         conn = getConnection();
         session = conn.createSession(false, Session.AUTO_ACKNOWLEDGE);

         Destination replyTo = m.getJMSReplyTo();
         MessageProducer producer = session.createProducer(replyTo);
         TextMessage reply = session.createTextMessage(result);

         producer.send(reply);
         producer.close();

      }
      catch(Exception e)
      {
         ctx.setRollbackOnly();
         e.printStackTrace();
         System.out.println("The Message Driven Bean failed!");
      }
      finally
      {
         if (conn != null)
         {
            try
            {
               closeConnection(conn);
            }
            catch(Exception e)
            {
               System.out.println("Could not close the connection!" +e);
            }
         }
      }
   }

   private String process(String s)
   {
      // flip the string

      String result = "";

      for (int i = 0; i < text.length(); i++)
      {
         result = text.charAt(i) + result;
      }
      return result;
   }

   public Connection getConnection() throws Exception
   {
      Connection connection = null;

      try
      {
         connection = cf.createConnection();
         connection.start();
      }
      catch(Exception e )
      {
         if(connection != null)
         {
            closeConnection(connection);
         }
         System.out.println("Failed to get connection...exception is " + e);
         throw e;
      }
```

```
      return connection;
   }

   public void closeConnection(Connection con) throws Exception
   {
      try
      {
         con.close();
      }
      catch(JMSException e)
      {
         System.out.println("Could not close connection " + con + " exception was " + e);
      }
   }

   public void ejbCreate()
   {
      try
      {
         InitialContext ic = new InitialContext();

         cf = (ConnectionFactory)ic.lookup("java:/JmsXA");

         ic.close();
      }
      catch(Exception e)
      {
         e.printStackTrace();
         throw new EJBException("Failure to get connection factory: " + e.getMessage());
      }
   }

   public void ejbRemove() throws EJBException
   {
      try
      {
         if(cf != null)
         {
            cf = null;
         }
      }
      catch(Exception e)
      {
         throw new EJBException("ejbRemove", e);
      }
   }

   public void setMessageDrivenContext(MessageDrivenContext ctx)
   {
      this.ctx = ctx;
   }


}
```

7.5. Using EJB3 Message Driven Beans

This example deploys a simple EJB3 Message Driven Bean that processes messages sent to a test queue. Once it receives a message and "processes" it, the MDB sends an acknowledgment message to a temporary destination cre-

ated by the sender for this purpose. The example is considered successful if the sender receives the acknowledgment message.

This example relies on having access to a running JBoss Messaging instance. The JBoss Messaging instance must be installed and started according to the "Installation" chapter of this document. The example will automatically deploy its own queue, unless a queue with the same name is already deployed.

This example also relies on having access to the `jboss-messaging-client.jar` archive that comes with the release bundle. If you run this example from an unzipped installation bundle, the example run script is correctly configured to find the client jar. Otherwise, you must modify example's `build.xml` accordingly.

The example was designed to deploy its server-side artifacts under a JBoss' `messaging` configuration. If you intend to use the script with a JBoss configuration that is named differently, please modify the example's `build.xml` accordingly.

Important

The JBoss instance that runs the Messaging server must also have EJB3 support previously installed. If the EJB3 support is not installed, the example will fail with an error message similar to:

```
C:\work\src\cvs\jboss-head\jms\docs\examples\ejb3mdb\build.xml:60: EJB3 does not seem
to be installed in C:\work\src\jboss-4.0.3-src\build\output\jboss-4.0.3/server/messaging!
Install it and try again.
```

For instructions on how to install EJB3 support, please go to JBoss EJB3 documentation page [http://docs.jboss.org/ejb3] or use the JBoss Installer.

The EJB3 Message Driven Bean source code:

```
@MessageDriven(activationConfig =
{
      @ActivationConfigProperty(propertyName="destinationType", propertyValue="javax.jms.Queue"),
      @ActivationConfigProperty(propertyName="destination", propertyValue="queue/testQueue"),
      @ActivationConfigProperty(propertyName="DLQMaxResent", propertyValue="10")
})
public class EJB3MDBExample implements MessageListener
{
   public void onMessage(Message m)
   {
      businessLogic(m);
   }

   private void businessLogic(Message m)
   {
      Connection conn = null;
      Session session = null;

      try
      {
         TextMessage tm = (TextMessage)m;

         String text = tm.getText();
         System.out.println("message " + text + " received");

         // flip the string
         String result = "";
```

```
         for(int i = 0; i < text.length(); i++)
         {
            result = text.charAt(i) + result;
         }

         System.out.println("message processed, result: " + result);


         InitialContext ic = new InitialContext();
         ConnectionFactory cf = (ConnectionFactory)ic.lookup("java:/JmsXA");
         ic.close();

         conn = cf.createConnection();
         conn.start();
         session = conn.createSession(false, Session.AUTO_ACKNOWLEDGE);

         Destination replyTo = m.getJMSReplyTo();
         MessageProducer producer = session.createProducer(replyTo);
         TextMessage reply = session.createTextMessage(result);

         producer.send(reply);
         producer.close();

      }
      catch(Exception e)
      {
         e.printStackTrace();
         System.out.println("The Message Driven Bean failed!");
      }
      finally
      {
         if (conn != null)
         {
            try
            {
               conn.close();
            }
            catch(Exception e)
            {
               System.out.println("Could not close the connection!" +e);
            }
         }
      }
   }
}
```

The basic test examples in this chapter serve as the sanity check for your JBoss Messaging installation. They also provide basic programming examples. To develop your own JMS services, you probably need to configure JBoss Messaging to setup your own destinations and connection factories etc. In Chapter 8, we will discuss JBoss Messaging configuration files and options.

8

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.

Starting with the JBoss Messaging 1.0.1 release, the service configuration is spread among several configuration files (the 1.0.0 release used to have all configuration information lumped together in the SAR's deployment descriptor `jboss-messaging.sar/META-INF/jboss-service.xml`). 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 9), `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.

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

</mbean>
```

8.1.1. We now discuss the MBean attributes of the ServerPeer MBean

8.1.1.1. PersistenceManager

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

8.1.1.2. 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.

8.1.1.3. JMSUserManager

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

8.1.1.4. 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.

8.1.1.5. 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.

8.1.1.6. ServerPeerID

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

8.1.1.7. DefaultQueueJNDIContext

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

8.1.1.8. DefaultTopicJNDIContext

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

8.1.1.9. Destinations

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

8.1.1.10. 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

8.1.1.11. 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)

8.1.1.12. 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)

8.1.1.13. 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

8.1.1.14. QueueStatsSamplePeriod

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

The default value is `10000` milliseconds

8.1.1.15. 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

8.1.1.16. MessageCounters

JBoss Messaging provides a message counter for each queue.

8.1.1.17. MessageCountersStatistics

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

8.1.1.18. 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.

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

8.1.2.1. 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

8.1.2.2. 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`

8.1.2.3. 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`

8.1.2.4. 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

8.1.2.5. 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`

8.1.2.6. 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`

8.1.2.7. ListMessageCountersHTML

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

8.1.2.8. ResetAllMesageCounters

This operation resets all message counters to zero.

8.1.2.9. ResetAllMesageCounters

This operation resets all message counter histories to zero.

8.1.2.10. EnableMessageCounters

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

8.1.2.11. DisableMessageCounters

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

8.1.2.12. RetrievePreparedTransactions

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

8.1.2.13. 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.

8.2. 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 `Data-Source` attribute in the persistence configuration file. Example data source configurations for each of the popular databases are available in the distribution.

8.3. 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

8.3.1. 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>
```

8.3.1.1. The non clustered post office has the following attributes

8.3.1.1.1. DataSource

The datasource the postoffice should use for persisting its mapping data

8.3.1.1.2. 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.

8.3.1.1.3. 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`

8.3.1.1.4. PostOfficeName

The name of the post office

8.3.2. 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">10000</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>
```

8.3.2.1. The nclustered post office has the following attributes

8.3.2.1.1. DataSource

The datasource the postoffice should use for persisting its mapping data

8.3.2.1.2. 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.

8.3.2.1.3. 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`

8.3.2.1.4. PostOfficeName

The name of the post office

8.3.2.1.5. NodeIDView

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

8.3.2.1.6. 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.

8.3.2.1.7. 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.

8.3.2.1.8. 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.

8.3.2.1.9. 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

8.3.2.1.10. CastTimeout

The maximum time to wait for a reply casting message synchronously

The default value is `5000` milliseconds

8.3.2.1.11. 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

8.3.2.1.12. 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

8.3.2.1.13. 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

8.3.2.1.14. 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

8.4. 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:

```
    <server>
```

```
      <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</deper
         <attribute name="DataSource">java:/DefaultDS</attribute>
         <attribute name="CreateTablesOnStartup">true</attribute>
         <attribute name="UsingBatchUpdates">false</attribute>
         <attribute name="MaxParams">500</attribute>
      </mbean>

      <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</deper
         <attribute name="PostOfficeName">JMS</attribute>
         <attribute name="DataSource">java:/DefaultDS</attribute>
         <attribute name="CreateTablesOnStartup">true</attribute>
      </mbean>

      <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</deper
         <attribute name="DataSource">java:/DefaultDS</attribute>
         <attribute name="CreateTablesOnStartup">true</attribute>
         <attribute name="SqlProperties"><![CDATA[
   POPULATE.TABLES.1=INSERT INTO JBM_USER (USER_ID,PASSWD,CLIENTID) VALUES ('dilbert','dogbert','dilbe
         ]]></attribute>
      </mbean>

   </server>
```

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

```
<server>

   <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</deper
         <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, TRANSACTIO
   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_ID)
   CREATE_IDX_MESSAGE_REF_SCHED_DELIVERY=CREATE INDEX JBM_MSG_REF_SCHED_DELIVERY ON JBM_MSG_REF (SCHED
   CREATE_MESSAGE=CREATE TABLE JBM_MSG (MESSAGE_ID BIGINT, RELIABLE CHAR(1), EXPIRATION BIGINT, TIMEST
   CREATE_TRANSACTION=CREATE TABLE JBM_TX (TRANSACTION_ID BIGINT, BRANCH_QUAL VARBINARY(254), FORMAT_I
   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, PAG
   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 CHANNE
      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_ID=?
      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 TRANSACTION_ID=
      LOAD_PAGED_REFS=SELECT MESSAGE_ID, DELIVERY_COUNT, PAGE_ORD, SCHED_DELIVERY FROM JBM_MSG_REF WHERE
      LOAD_UNPAGED_REFS=SELECT MESSAGE_ID, DELIVERY_COUNT, SCHED_DELIVERY FROM JBM_MSG_REF WHERE STATE =
      LOAD_REFS=SELECT MESSAGE_ID, DELIVERY_COUNT, SCHED_DELIVERY FROM JBM_MSG_REF WHERE STATE = 'C' AND
      UPDATE_REFS_NOT_PAGED=UPDATE JBM_MSG_REF SET PAGE_ORD = NULL WHERE PAGE_ORD BETWEEN ? AND ? AND CHA
      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 MESSAGE_ID
      UPDATE_CHANNEL_ID=UPDATE JBM_MSG_REF SET CHANNEL_ID = ? WHERE CHANNEL_ID = ?
      LOAD_MESSAGES=SELECT MESSAGE_ID, RELIABLE, EXPIRATION, TIMESTAMP, PRIORITY, HEADERS, PAYLOAD, TYPE
      INSERT_MESSAGE=INSERT INTO JBM_MSG (MESSAGE_ID, RELIABLE, EXPIRATION, TIMESTAMP, PRIORITY, HEADERS,
      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) VALUES(
      DELETE_TRANSACTION=DELETE FROM JBM_TX WHERE TRANSACTION_ID = ?
      SELECT_PREPARED_TRANSACTIONS=SELECT TRANSACTION_ID, BRANCH_QUAL, FORMAT_ID, GLOBAL_TXID FROM JBM_TX
      SELECT_MESSAGE_ID_FOR_REF=SELECT MESSAGE_ID, CHANNEL_ID FROM JBM_MSG_REF WHERE TRANSACTION_ID = ? A
      SELECT_MESSAGE_ID_FOR_ACK=SELECT MESSAGE_ID, CHANNEL_ID FROM JBM_MSG_REF WHERE TRANSACTION_ID = ? A
      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>

      <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, QU
   INSERT_BINDING=INSERT INTO JBM_POSTOFFICE (POSTOFFICE_NAME, NODE_ID, QUEUE_NAME, COND, SELECTOR, CHANN
   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 WH
         ]]></attribute>
      </mbean>

      <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, CL
   CREATE_ROLE_TABLE=CREATE TABLE JBM_ROLE (ROLE_ID VARCHAR(32) NOT NULL, USER_ID VARCHAR(32) NOT NULL, P
   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-
         ]]></attribute>
      </mbean>
```

```
</server>
```

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

8.4.1.1. 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`

8.4.1.2. 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`

8.4.1.3. 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`

8.4.1.4. 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`

8.4.1.5. 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.

8.4.1.6. 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`

8.5. 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>
```

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

8.5.1.1. 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`

8.5.1.2. 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`

8.5.1.3. 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.

8.6. Configuring Destinations

8.6.1. 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>
....
```

8.6.2. Configuring queues

8.6.2.1. We now discuss the MBean attributes of the Queue MBean

8.6.2.1.1. Name

The name of the queue

8.6.2.1.2. JNDIName

The JNDI name where the queue is bound

8.6.2.1.3. DLQ

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

8.6.2.1.4. ExpiryQueue

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

8.6.2.1.5. RedeliveryDelay

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

8.6.2.1.6. 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.

8.6.2.1.7. 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 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.

8.6.2.1.8. CreatedProgrammatically

Returns `true` if the queue was created programmatically

8.6.2.1.9. MessageCount

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

8.6.2.1.10. ScheduledMessageCount

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

8.6.2.1.11. 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.

8.6.2.1.12. Clustered

Clustered destinations must have this set to `true`

8.6.2.1.13. MessageCounter

Each queue maintains a message counter.

8.6.2.1.14. MessageCounterStatistics

The statistics for the message counter

8.6.2.1.15. MessageCounterHistoryDayLimit

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

8.6.2.1.16. ConsumerCount

The number of consumers currently consuming from the queue.

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

8.6.2.2.1. RemoveAllMessages

Remove (and delete) all messages from the queue.

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

8.6.2.2.2. 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

8.6.2.2.3. 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

8.6.2.2.4. 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

8.6.2.2.5. ResetMessageCounter

Resets the message counter to zero.

8.6.2.2.6. ResetMessageCounterHistory

Resets the message counter history.

8.6.2.2.7. ListMessageCounterAsHTML

Lists the message counter in an easy to display HTML format

8.6.2.2.8. ListMessageCounterHistoryAsHTML

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

8.6.3. Configuring topics

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

8.6.3.1.1. Name

The name of the topic

8.6.3.1.2. JNDIName

The JNDI name where the topic is bound

8.6.3.1.3. DLQ

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

8.6.3.1.4. ExpiryQueue

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

8.6.3.1.5. RedeliveryDelay

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

8.6.3.1.6. 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.

8.6.3.1.7. 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 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.

8.6.3.1.8. CreatedProgrammatically

Returns `true` if the topic was created programmatically

8.6.3.1.9. 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.

8.6.3.1.10. Clustered

Clustered destinations must have this set to `true`

8.6.3.1.11. MessageCounterHistoryDayLimit

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

8.6.3.1.12. MessageCounters

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

8.6.3.1.13. AllMessageCount

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

8.6.3.1.14. DurableMessageCount

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

8.6.3.1.15. NonDurableMessageCount

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

8.6.3.1.16. AllSubscriptionsCount

The count of all subscriptions on this topic

8.6.3.1.17. DurableSubscriptionsCount

The count of all durable subscriptions on this topic

8.6.3.1.18. NonDurableSubscriptionsCount

The count of all non durable subscriptions on this topic

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

8.6.3.2.1. 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

8.6.3.2.2. ListAllSubscriptions

List all subscriptions of this topic

8.6.3.2.3. ListDurableSubscriptions

List all durable subscriptions of this topic

8.6.3.2.4. ListNonDurableSubscriptions

List all non durable subscriptions of this topic

8.6.3.2.5. ListAllSubscriptionsAsHTML

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

8.6.3.2.6. ListDurableSubscriptionsAsHTML

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

8.6.3.2.7. ListNonDurableSubscriptionsAsHTML

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

8.6.3.2.8. 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 specifying the selector you can limit the messages returned.

8.6.3.2.9. 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 specifying the selector you can limit the messages returned.

8.6.3.2.10. 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 specifying the selector you can limit the messages returned.

8.6.4. Deploying a new destination

For a JBoss 4.0.x installation, JBoss Messaging is deployed in its own class loading domain. Because of that you need to deploy a new destinations to use with JBoss Messaging within the same class loading domain.

To deploy a new destination, create a new deployment descriptor named `myqueue-service.xml` (or anything else that ends in `-service.xml`) and copy it to the JBoss instance deployment directory `$JBOSS_HOME/server/messaging/deploy`.

An example of a scoped destination deployment descriptor is listed below:

```
<?xml version="1.0" encoding="UTF-8"?>
<server>
   <loader-repository>jboss.messaging:loader=ScopedLoaderRepository
   <loader-repository-config>java2ParentDelegation=false</loader-repository-config>
   </loader-repository>
   <mbean
      code="org.jboss.jms.server.destination.Queue"
     name="jboss.messaging.destination:service=Queue,name=testQueue"
     xmbean-dd="xmdesc/Queue-xmbean.xml">
      <depends optional-attribute-name="ServerPeer">
         jboss.messaging:service=ServerPeer
```

```
      </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>
      <attribute name="fullSize">75000</attribute>
      <attribute name="pageSize">2000</attribute>
      <attribute name="downCacheSize">2000</attribute>
   </mbean>
</server>
```

8.7. Configuring Connection Factories

With the default configuration JBoss Messaging binds just one connection factory in JNDI at start-up. This connection factory has no client ID and is bound into the following JNDI contexts: `/ConnectionFactory`, `/XAConnectionFactory`, `java:/ConnectionFactory`, `java:/XAConnectionFactory`

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, or if you want different connection factories to use different transports. 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 either be clustered or non clustered. Clustered connection factories create subsequent connections on different nodes of the cluster according to the load balancing policy. The default load balancing policy is round robin.

An example connection factory configuration is presented below:

```
<server>
   <loader-repository>
      jboss.messaging:loader=ScopedLoaderRepository
         <loader-repository-config>
            java2ParentDelegation=false
         </loader-repository-config>
   </loader-repository>
   <mbean
      code="org.jboss.jms.server.connectionfactory.ConnectionFactory"
      name="jboss.messaging.destination:service=ConnectionFactory"
      xmbean-dd="xmdesc/ConnectionFactory-xmbean.xml">
      <constructor>
         <arg type="java.lang.String" value="myClientID"/>
      </constructor>
      <depends optional-attribute-name="ServerPeer">
         jboss.messaging:service=ServerPeer
      </depends>
      <depends optional-attribute-name="Connector">
         jboss.messaging:service=Connector,transport=socket
      </depends>
      <attribute name="PrefetchSize">10</attribute>
```

```
      <attribute name="DefaultTempQueueFullSize">1000</attribute>
      <attribute name="DefaultTempQueuePageSize">50</attribute>
      <attribute name="DefaultTempQueueDownCacheSize">50</attribute>
      <attribute name="JNDIBindings">
         <bindings>
            <binding>/MyConnectionFactory1</binding>
            <binding>/factories/cf1</binding>>
         </bindings>
      </attribute>
   </mbean>
</server>
```

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 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.

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

8.7.1.1. ClientID

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

8.7.1.2. JNDIBindings

Returns a list of the JNDI bindings for this connection factory

8.7.1.3. 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.

8.7.1.4. 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.

8.7.1.5. 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`

8.7.1.6. 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.

8.8. 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

8.9. 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

8.10. 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.

9

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 cluster router policy
- Choosing the message redistributon policy

9.1. 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
```

9.2. 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.

10

Generating Performance Benchmark Results

As we discussed in Chapter 1, the key advantage of JBoss Messaging is its superior performance. In fact, the JBoss Messaging comes with a set of standard performance test. You can run them on your server and generate your own performance benchmark results. In this chapter, we will show you how to run a JBoss Messaging server and a JBossMQ server side-by-side on a single machine, and compare their performance. To get the performance tests, you have to obtain the JBoss Messaging source code from SVN as described in Chapter 4.

To get the performance tests, you first need to check out the source code and build the project. The location of the framework's source code will be referred to as $PERF_HOME throughout the rest of the document.

```
svn co https://svn.jboss.org/repos/messaging/projects/perf/trunk messaging-perf
cd $PERF_HOME
ant
```

The test consists in sending bursts of 1000 0 Kilobytes non-persistent messages to both JBoss Messaging and JBossMQ instances while gradually increasing the send rate (200 messages/sec, 400 messages/sec, etc) and measuring the receive rate. At the end, the framework generates the graph representing the receive rate as function of the send rate for two executions (JBoss Messaging and JBossMQ).

10.1. Run JBoss Messaging and JBossMQ Side-by-side

To run performance tests side-by-side on the same machine, we assume that you create two JBoss AS configurations with the JBoss Messaging and JBossMQ modules respectively. We assume that the JBoss Messaging module is installed in the `server/messaging` directory (see Chapter 5), and the default JBossMQ module is installed in `server/jbossmq` directory (just copy the original `default` directory that comes with the server).

Now, if you run the two configurations on the same server, there will be port conflicts. To avoid that, we use the JBoss `ServiceBindingManager` to increase the port numbers in the `jbossmq` configuration by 100 (i.e., the JNDI service will be available at port 1199 instead of 1099). To do that, un-comment the following line in `server/jbossmq/conf/jboss-service.xml`

```
<mbean code="org.jboss.services.binding.ServiceBindingManager"
       name="jboss.system:service=ServiceBindingManager">

  <attribute name="ServerName">ports-01</attribute>WWWr
  <attribute name="StoreURL">
    ../docs/examples/binding-manager/sample-bindings.xml
  </attribute>
  <attribute name="StoreFactoryClassName">
    org.jboss.services.binding.XMLServicesStoreFactory
  </attribute>
</mbean>
```

```
```

Now, you can start the `messaging` and `jbossmq` configurations side-by-side for testing.

```
run -c messaging
run -c jbossmq
```

10.2. Setup the Tests

The performance framework relies on distributed executors to send messages into the providers being tested. The executors can run standalone in their own VM and act as "remote" JMS connections, or colocated, in which case they are deployed as JBoss services and simulate "colocated" JMS connections.

In order to correctly deploy the colocated executors, the framework relies on the `JBOSS_HOME` environment variable. It assumes directories `$JBOSS_HOME/server/messaging` and `$JBOSS_HOME/server/jbossmq` exist.

```
ant sar
ant start-executors
```

Next, we need to deploy test message destinations. They are in the `messaging-destinations-service.xml`, `jbossmq-destinations-service.xml` files. Feel free to add your own destinations (must add equivalent ones in both files) and then deploy them via the following command.

```
ant deploy-destinations
```

10.3. Configure Test Runs

The `etc/perf.xml` file is used to configure tests. In our setting (i.e., `jbossmq` runs in +100 port range from default), the `<providers>` section should look like the following. We can easily run the two JMS server configurations on different machines or in other port ranges. You just need to change the host and port numbers here for tests.

```
<provider name="JBossMessaging">
  <factory>org.jnp.interfaces.NamingContextFactory</factory>
  <url>jnp://localhost:1099</url>
  <pkg>org.jboss.naming:org.jnp.interfaces</pkg>
  <executor name="REMOTE" url="rmi://localhost:7777/standalone"/>
  <executor name="REMOTE2" url="rmi://localhost:7777/standalone2"/>
  <executor name="COLOCATED" url="rmi://localhost:7777/local-messaging"/>
  <executor name="COLOCATED2" url="rmi://localhost:7777/local-messaging2"/>
</provider>

<provider name="JBossMQ">
  <factory>org.jnp.interfaces.NamingContextFactory</factory>
  <url>jnp://localhost:1199</url>
  <pkg>org.jboss.naming:org.jnp.interfaces</pkg>
  <executor name="REMOTE" url="rmi://localhost:7777/standalone"/>
```

```
  <executor name="REMOTE2" url="rmi://localhost:7777/standalone2"/>
  <executor name="COLOCATED" url="rmi://localhost:7777/local-jbossmq"/>
  <executor name="COLOCATED2" url="rmi://localhost:7777/local-jbossmq2"/>
</provider>
```

The performance configuration section configures how to ramp up the load from 200 messages / sec to 3000 message / sec. We will gather statistics on the number of processed messages versus the number of sent messages.

```
<performance-test name="Throughput 0 KB Message
   Non-Persistent Non-Transactional, 1 sender, 1 receiver">

  <message-size>0</message-size>
  <messages>10000</messages>

  <drain/>

  <parallel>
    <send rate="200" executor="COLOCATED"/>
    <receive executor="COLOCATED2"/>
  </parallel>

  <parallel>
    <send rate="400" executor="COLOCATED"/>
    <receive executor="COLOCATED2"/>
  </parallel>

  <parallel>
    <send rate="800" executor="COLOCATED"/>
    <receive executor="COLOCATED2"/>
  </parallel>

  <parallel>
    <send rate="1000" executor="COLOCATED"/>
    <receive executor="COLOCATED2"/>
  </parallel>

  <parallel>
    <send rate="1500" executor="COLOCATED"/>
    <receive executor="COLOCATED2"/>
  </parallel>

  <parallel>
    <send rate="2000" executor="COLOCATED"/>
    <receive executor="COLOCATED2"/>
  </parallel>

  <parallel>
    <send rate="2500" executor="COLOCATED"/>
    <receive executor="COLOCATED2"/>
  </parallel>

  <parallel>
    <send rate="3000" executor="COLOCATED"/>
    <receive executor="COLOCATED2"/>
  </parallel>

  <execution provider="JBossMessaging"/>
  <execution provider="JBossMQ"/>

</performance-test>
```

10.4. Run the Tests

To run the tests, simply execute `ant run` from the command line. You can access the benchmark result graphs from `output/results/benchmark-results.html`.

After running the test, you can clean up the executors and test destinations using the following commands.

```
ant stop-executors
ant undeploy-destinations
```