JBoss Messaging 1.4 User's Guide

Enterprise Messaging from JBoss

Table of Contents

1. About JBoss Messaging 1.4.0. 1
2. Introduction 2
    2.1. JBoss Messaging support cover from Red Hat 2
    2.2. JBoss Messaging Features 2
    2.3. Compatibility with JBossMQ 4
3. Download Software 5
    3.1. SVN Access 5
4. JBoss Messaging Installation 6
    4.1. Installing JBoss Messaging on JBoss AS 4.2 6
        4.1.1. Automated Installation 6
        4.1.2. Manual Installation 9
        4.1.3. Extra steps to complete your installation 12
    4.2. Starting the Server 14
    4.3. Installation Validation 16
    4.4. Accessing JBoss Messaging from a remote client 17
5. Running the Examples 19
6. Configuration 21
    6.1. Configuring the ServerPeer 21
        6.1.1. ServerPeer attributes 23
            6.1.1.1. ServerPeerID 23
            6.1.1.2. DefaultQueueJNDIContext 23
            6.1.1.3. DefaultTopicJNDIContext 23
            6.1.1.4. PostOffice 23
            6.1.1.5. SecurityDomain 23
            6.1.1.6. DefaultSecurityConfig 23
            6.1.1.7. DefaultDLQ 24
            6.1.1.8. DefaultMaxDeliveryAttempts 24
            6.1.1.9. DefaultExpiryQueue 24
            6.1.1.10. DefaultRedeliveryDelay 24
            6.1.1.11. MessageCounterSamplePeriod 24
            6.1.1.12. FailoverStartTimeout 24
            6.1.1.13. FailoverCompleteTimeout 24
            6.1.1.14. DefaultMessageCounterHistoryDayLimit 25
            6.1.1.15. ClusterPullConnectionFactory 25
            6.1.1.16. DefaultPreserveOrdering 25
            6.1.1.17. RecoverDeliveriesTimeout 25
            6.1.1.18. SuckerPassword 25
            6.1.1.19. StrictTCK 25
            6.1.1.20. Destinations 25
            6.1.1.21. MessageCounters 25
            6.1.1.22. MessageCountersStatistics 26
            6.1.1.23. SupportsFailover 26
            6.1.1.24. PersistenceManager 26
            6.1.1.25. JMSUserManager 26

6.1.1.26. We now discuss the MBean operations of the ServerPeer MBean. 26
6.2. Changing the Database 28
6.3. Configuring the Post office 28
6.3.1. The post office has the following attributes 31
6.3.1.1. DataSource 31
6.3.1.2. SQLProperties 31
6.3.1.3. CreateTablesOnStartup 31
6.3.1.4. PostOfficeName 31
6.3.1.5. NodeIDView 31
6.3.1.6. GroupName 31
6.3.1.7. Clustered 32
6.3.1.8. StateTimeout 32
6.3.1.9. CastTimeout 32
6.3.1.10. MaxConcurrentReplications 32
6.3.1.11. ControlChannelConfig 32
6.3.1.12. DataChannelConfig 32
6.4. Configuring the Persistence Manager 32
6.4.1. We now discuss the MBean attributes of the PersistenceManager MBean 34
6.4.1.1. CreateTablesOnStartup 34
6.4.1.2. UsingBatchUpdates 35
6.4.1.3. UsingBinaryStream 35
6.4.1.4. UsingTrailingByte 35
6.4.1.5. SupportsBlobOnSelect 35
6.4.1.6. SQLProperties 35
6.4.1.7. MaxParams 35
6.5. Configuring the JMS user manager 36
6.5.1. We now discuss the MBean attributes of the JMSUserManager MBean 36
6.5.1.1. CreateTablesOnStartup 36
6.5.1.2. UsingBatchUpdates 36
6.5.1.3. SQLProperties 36
6.6. Configuring Destinations 37
6.6.1. Pre-configured destinations 37
6.6.2. Configuring queues 39
6.6.2.1. We now discuss the attributes of the Queue MBean 39
6.6.2.2. We now discuss the MBean operations of the Queue MBean 42
6.6.3. Configuring topics 43
6.6.3.1. We now discuss the MBean attributes of the Topic MBean 43
6.6.3.2. We now discuss the MBean operations of the Topic MBean 45
6.7. Configuring Connection Factories 46
6.7.1. We now discuss the MBean attributes of the ConnectionFactory MBean 48
6.7.1.1. ClientID 48
6.7.1.2. JNDIBindings 48
6.7.1.3. PrefetchSize 48
6.7.1.4. SlowConsumers 48
6.7.1.5. StrictTck 48
6.7.1.6. Temporary queue paging parameters 48
6.7.1.7. DupsOKBatchSize 48
6.7.1.8. SupportsLoadBalancing 48
6.7.1.9. SupportsFailover 49

6.7.1.10. DisableRemotingChecks 49
6.7.1.11. LoadBalancingFactory 49
6.7.1.12. Connector 49
6.8. Configuring the remoting connector 50
6.9. ServiceBindingManager 52
7. JBoss Messaging Clustering Notes 53
7.1. Unique server peer id 53
7.2. Clustered destinations 53
7.3. Clustered durable subs 53
7.4. Clustered temporary destinations 53
7.5. Non clustered servers 53
7.6. Message ordering in the cluster 54
7.7. Idempotent operations 54
7.8. Clustered connection factories 54
8. JBoss Messaging XA Recovery Configuration 55
9. JBoss Messaging Message Bridge Configuration 57
9.1. Message bridge overview 57
9.2. Bridge deployment 58
9.3. Bridge configuration 58
9.3.1. SourceProviderLoader 60
9.3.2. TargetProviderLoader 60
9.3.3. SourceDestinationLookup 60
9.3.4. TargetDestinationLookup 60
9.3.5. SourceUsername 61
9.3.6. SourcePassword 61
9.3.7. TargetUsername 61
9.3.8. TargetPassword 61
9.3.9. QualityOfServiceMode 61
9.3.10. Selector 61
9.3.11. MaxBatchSize 61
9.3.12. MaxBatchTime 62
9.3.13. SubName 62
9.3.14. ClientID 62
9.3.15. FailureRetryInterval 62
9.3.16. MaxRetries 62
9.3.17. AddMessageIDInHeader 62

1

About JBoss Messaging 1.4.0.

JBoss Messaging is the new enterprise messaging system from JBoss. It is a complete rewrite of JBossMQ, the legacy JBoss JMS provider.

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

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

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

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

JBoss Messaging can be easily installed in JBoss Application Server 4.2 using a few simple steps to remove JBoss MQ and replace with JBoss Messaging.

Once JBoss Messaging becomes the default JMS provider in JBoss Application Server, there will be no need to do any manual installation.

From release 1.4.0. onwards JBoss Messaging is designed for JBoss 4.2 only and is Java 5.

The procedure of installing JBoss Messaging into JBoss Application Server is detailed in this guide.

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

Permanent Team: Tim Fox (Project Lead), Jeff Mesnil (Core Developer), Andy Taylor (Core Developer), Clebert Suconic (Core Developer)

Contributors: Ovidiu Feodorov (Project Founder), Sergey Koshcheyev, Ron Sigal, Madhu Konda, Jay Howell, Tyronne Wickramarathne, Aaron Walker, Adrian Brock, Rajdeep Dua, Tom Elrod, Alex Fu, Juha Lindfors, Alexey Loubyansky, Luc Texier, Scott Stark, Aslak Knutsen

Messaging support team: Jay Howell, David Boeren, Mike Clark, Tyronne Wickramarathne

Other thanks to: Mark Little and Pete Bennett

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 performant clustering implementation.

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

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

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

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

2.1. JBoss Messaging support cover from Red Hat

JBoss Messaging is destined to become part of both Application Server Platform and Service Integration Platform as default JMS provider.Production support will then be fully available for these plaforms and it will cover JBoss Messaging.

There is a currently an early adopters program where production support is given for JBoss Messaging. Applications are considered on a case-by-case basis.

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

2.2. JBoss Messaging Features

JBoss Messaging provides:

- A fully compatible and Sun certified JMS 1.1 implementation, that currently works with a standard 4.2 or later JBoss Application Server installation.
- A strong focus on performance, reliability and scalability with high throughput and low latency.
- A foundation for JBoss ESB for SOA initiatives; JBoss ESB uses JBoss Messaging as its default JMS provider.

Other JBoss Messaging features include:

- Publish-subscribe and point-to-point messaging models
- Persistent and non-persistent messages
- Guaranteed message delivery that ensures that messages arrive once and only once where required
- Transactional and reliable - supporting ACID semantics
- Customizable security framework based on JAAS
- Fully integrated with JBoss Transactions (formerly known as Arjuna JTA) for full transaction recoverability.
- Extensive JMX management interface
- Support for most major databases including Oracle, Sybase, MS SQL Server, PostgreSQL and MySQL
- HTTP transport to allow use through firewalls that only allow HTTP traffic
- SSL transport
- Configurable DLQs (Dead Letter Queues) and Expiry Queues
- Message statistics. Gives you a rolling historical view of what messages were delivered to what queues and subscriptions
- Automatic paging of messages to storage. Allows the use of very large queues - too large to fit in memory at once

Clustering features:

- Fully clustered queues and topics. "Logical" queues and topics are distributed across the cluster. You can send to a queue or a topic from any node, and receive from any other.
- Fully clustered durable subscriptions. A particular durable subscription can be accessed from any node of the cluster - allowing you to spread processing load from that subscription across the cluster.
- Fully clustered temporary queues. Send a message with a replyTo of a temp queue and it can be sent back on any node of the cluster.
- Intelligent message redistribution. Messages are automatically moved between different nodes of the cluster if consumers are faster on one node than another. This can help prevent starvation or build up of messages on particular nodes.

- Message order protection. If you want to ensure that the order of messages produced by a producer is the same as is consumed by a consumer then you can set this to true. This works even in the presence of message redistribution.
- Fully transparent failover. When a server fails, your sessions continue without exceptions on a new node as if nothing happened. (Fully configurable - If you don't want this you can fall back to exceptions being thrown and manually recreation of connections on another node)
- High availability and seamless fail-over. If the node you are connected to fails, you will automatically fail over to another node and will not lose any persistent messages. You can carry on with your session seamlessly where you left off. Once and only once delivery of persistent messages is respected at all times.
- Message bridge. JBoss Messaging contains a message bridge component which enables you to bridge messages between any two JMS1.1 destinations on the same or physical separate locations. (E.g. separated by a WAN). This allows you to connect geographically separate clusters, forming huge globally distributed logical queues and topics.

2.3. Compatibility with JBossMQ

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

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

Important

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

3

Download Software

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

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

3.1. 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.

4

JBoss Messaging Installation

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

By default, JBoss AS 4 ships with JBossMQ as default JMS provider. In order to use the JBoss AS instance with JBoss Messaging, you need to perform the installation procedure described below.

Note

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

Note

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

Note

JBoss Messaging is built against the JBoss AS 4.2 libraries which are built using Java 5. Therefore JBoss Messaging only runs with Java 5 or later.

4.1. Installing JBoss Messaging on JBoss AS 4.2

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

- If you have a completely clean JBoss AS 4.2.0 installation then you can do an Section 4.1.1 .
- If you have a JBoss 4.2.0 that you have changed in some way from a clean JBoss AS 4.2.0 installation then you will need to do a Section 4.1.2 .

4.1.1. Automated Installation

Note

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

Messaging installation

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

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

- Set up the `JBOSS_HOME` environment variable to point to the JBoss 4.2 installation you want to use JBoss Messaging with.

  Run the installation script, available in the `util` directory of the release bundle as follows:

  If you want to create a simple non clustered installion based on the default configuration:

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

  If you want to create a clustered installation based on the all configuration or change the configuration name:

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

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

  The messaging.config.source variable determines which JBoss AS configuration (e.g. default or all) to base the installation on

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

- **Warning**

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

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

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

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

- Remove hsqldb-ds.xml from `$JBOSS_CONFIG/deploy`
- Copy your database driver to `$JBOSS_CONFIG/lib`

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

- Ensure the `ServerPeerID` MBean attribute value in messaging-service.xml is unique for each node. The `ServerPeerID` value must be a valid integer. Every node MUST have a unique id, including those just connected by message bridges.

Warning

Each node must have a unique `ServerPeerID` irrespective of whether you are using clustering.

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

  - Uncomment binding manager service from $JBOSS_CONFIG/conf/jboss-service.xml
  - Specify the desired port rage (e.g. ports-01, ports-02... etc)
  - Look at $JBOSS_HOME/docs/examples/binding-manager/sample-bindings.xml. Here is an example:

```
<service-config name="jboss.messaging:service=Connector,transport=bisocket"
                delegateClass="org.jboss.services.binding.AttributeMappingDelegate">
   <delegate-config>
      <attribute name="Configuration"><![CDATA[
          <config>
      <invoker transport="bisocket">

         <!-- There should be no reason to change these parameters - warning!
              Changing them may stop JBoss Messaging working correctly -->
         <attribute name="marshaller" isParam="true">org.jboss.jms.wireformat.JMSWireFormat</attrib
         <attribute name="unmarshaller" isParam="true">org.jboss.jms.wireformat.JMSWireFormat</attr
         <attribute name="dataType" isParam="true">jms</attribute>
         <attribute name="socket.check_connection" isParam="true">false</attribute>
         <attribute name="timeout" isParam="true">0</attribute>
         <attribute name="serverBindAddress">${jboss.bind.address}</attribute>
         <attribute name="serverBindPort">4657</attribute>
         <attribute name="clientSocketClass" isParam="true">org.jboss.jms.client.remoting.ClientSoc
         <attribute name="serverSocketClass" isParam="true">org.jboss.jms.server.remoting.ServerSoc
         <attribute name="numberOfCallRetries" isParam="true">1</attribute>
         <attribute name="pingFrequency" isParam="true">214748364</attribute>
         <attribute name="pingWindowFactor" isParam="true">10</attribute>
         <attribute name="onewayThreadPool">org.jboss.jms.server.remoting.DirectThreadPool</attribu

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

         <!-- Number of seconds to wait for a connection in the client pool to become free -->
         <attribute name="numberOfRetries" isParam="true">10</attribute>

         <!-- Max Number of connections in client pool. This should be significantly higher than
              the max number of sessions/consumers you expect -->
```

```
               <attribute name="JBM_clientMaxPoolSize" isParam="true">200</attribute>

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

            </invoker>
            <handlers>
               <handler subsystem="JMS">org.jboss.jms.server.remoting.JMSServerInvocationHandler</handler
            </handlers>
         </config>
            ]]></attribute>
         </delegate-config>
         <binding port="4657"/>
      </service-config>
```

Warning

You must ensure that the config (like above) is identical to that in `remoting-bisocket-service.xml` With the exception of the actual serverBindPort which clearly must be different for each ports range. Please note that the default JBoss Messaging service binding manager bindings in `sample-bindings.xml` shipped with JBAS 4.2.0 may be out of date and you will need to copy the config from `remoting-bisocket-service.xml. DO NOT just copy and paste from the above example - copy it from the JBoss Messaging distribution.`

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

- There are few extra steps at Section 4.1.3
- That's it

4.1.2. Manual Installation

Note

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

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

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

- Move `$JBOSS_CONFIG/deploy/jms/hajndi-jms-ds.xml` and `$JBOSS_CONFIG/deploy/jms/jms-ra.rar` to `$JBOSS_CONFIG/deploy`

- Remove the old JBoss MQ by removing the directory `$JBOSS_CONFIG/deploy/jms.`
  .

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

  Make sure you don't have any JBossMQ files under `$JBOSS_CONFIG/deploy-hasingleton`. For that just remove `$JBOSS_CONFIG/deploy-hasingleton/jms`

- Add a security policy called "messaging" on $JBOSS_CONFIG/config/login-config.xml. You could use this as an example, or create one according to JBoss Security Documentation:

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

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

Note

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

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

```
# messaging-users.properties

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

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

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

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

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

- **Warning**

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

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

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

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

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

  Warning

  Each node must have a unique `ServerPeerID` irrespective of whether you are using clustering.

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

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

```
<service-config name="jboss.messaging:service=Connector,transport=bisocket"
                delegateClass="org.jboss.services.binding.AttributeMappingDelegate">
   <delegate-config>
      <attribute name="Configuration"><![CDATA[
          <config>
      <invoker transport="bisocket">

         <!-- There should be no reason to change these parameters - warning!
              Changing them may stop JBoss Messaging working correctly -->
```

```
            <attribute name="marshaller" isParam="true">org.jboss.jms.wireformat.JMSWireFormat</attrib
            <attribute name="unmarshaller" isParam="true">org.jboss.jms.wireformat.JMSWireFormat</attr
            <attribute name="dataType" isParam="true">jms</attribute>
            <attribute name="socket.check_connection" isParam="true">false</attribute>
            <attribute name="timeout" isParam="true">0</attribute>
            <attribute name="serverBindAddress">${jboss.bind.address}</attribute>
            <attribute name="serverBindPort">4657</attribute>
            <attribute name="clientSocketClass" isParam="true">org.jboss.jms.client.remoting.ClientSoc
            <attribute name="serverSocketClass" isParam="true">org.jboss.jms.server.remoting.ServerSoc
            <attribute name="numberOfCallRetries" isParam="true">1</attribute>
            <attribute name="pingFrequency" isParam="true">214748364</attribute>
            <attribute name="pingWindowFactor" isParam="true">10</attribute>
            <attribute name="onewayThreadPool">org.jboss.jms.server.remoting.DirectThreadPool</attribu

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

            <!-- Number of seconds to wait for a connection in the client pool to become free -->
            <attribute name="numberOfRetries" isParam="true">10</attribute>

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

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

         </invoker>
         <handlers>
            <handler subsystem="JMS">org.jboss.jms.server.remoting.JMSServerInvocationHandler</handler
         </handlers>
      </config>
         ]]></attribute>
      </delegate-config>
      <binding port="4657"/>
   </service-config>
```

Warning

You must ensure that the config (like above) is identical to that in `remoting-bisocket-service.xml` With the exception of the actual serverBindPort which clearly must be different for each ports range. Please note that the default JBoss Messaging service binding manager bindings in `sample-bindings.xml` shipped with JBAS 4.2.0 may be out of date and you will need to copy the config from `remoting-bisocket-service.xml. DO NOT just copy and paste from the above example - copy it from the JBoss Messaging distribution.`

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

- There are few extra steps at Section 4.1.3
- That's it

4.1.3. Extra steps to complete your installation

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

- **Warning**

  SECURITY RISK! To avoid a security risk, you MUST specify the value of the attribute SuckerPassword in the Server Peer config (messaging-service.xml). If you do not specify a value, the default value will be used. Any person that knows the default value will be able to access to all destinations on the server. The password chosen should only be exposed to administrators

- **Note**

  JBoss Messaging 1.4.0 requires a patched version of jboss-remoting.jar. This version is available on JBoss EAP 4.3, so if you're using a different version of JBoss you should download the patched version which can be found here [http://repository.jboss.com/jboss/remoting/2.2.2.SP4-brew/]The version is JBoss Remoting 2.2.2.SP4-brew. Please download it and copy it into the `$JBOSS_HOME/server/<your server name>/lib directory` of any server profiles that use JBoss Messaging 1.4.0. If you are using JBoss Messaging from a standalone client also make sure this jar is on your classpath *before* jbossall-client.jar.

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

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

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

  Make sure it looks like this:

  ```
  <CreateJBossMQDestination>false</CreateJBossMQDestination>
  ```

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

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

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

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

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

```
### Remove these lines:

<!-- Security domain for JBossMQ -->
<application-policy name = "jbossmq">
```

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

<!-- Security domain for JBossMQ when using file-state-service.xml
<application-policy name = "jbossmq">
 <authentication>
    <login-module code = "org.jboss.mq.sm.file.DynamicLoginModule"
       flag = "required">
       <module-option name = "unauthenticatedIdentity">guest</module-option>
       <module-option name = "sm.objectname">jboss.mq:service=StateManager</module-option>
    </login-module>
 </authentication>
</application-policy>
-->
```

4.2. 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 <config name>
```

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

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

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

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

13:19:15,166 INFO  [ServerPeer] JBoss Messaging 1.4.0.GA server [0] started
13:19:15,411 INFO  [ConnectionFactory] Connector bisocket://127.0.0.1:4457 has leasing enabled
, lease period 10000 milliseconds
13:19:15,412 INFO  [ConnectionFactory] [/ConnectionFactory, /XAConnectionFactory, java:/Connec
tionFactory, java:/XAConnectionFactory] started
```

```
13:19:15,412 WARN  [ConnectionFactoryJNDIMapper] supportsFailover attribute is true on connect
ion factory: jboss.messaging.connectionfactory:service=ClusteredConnectionFactory but
post office is non clustered. So connection factory will *not* support failover
13:19:15,413 WARN  [ConnectionFactoryJNDIMapper] supportsLoadBalancing attribute is true on co
nnection factory: jboss.messaging.connectionfactory:service=ClusteredConnectionFactory
but post office is non clustered. So connection factory will *not* support load balanc
ing
13:19:15,449 INFO  [ConnectionFactory] Connector bisocket://127.0.0.1:4457 has leasing enabled
, lease period 10000 milliseconds
13:19:15,449 INFO  [ConnectionFactory] [/ClusteredConnectionFactory, /ClusteredXAConnectionFac
tory, java:/ClusteredConnectionFactory, java:/ClusteredXAConnectionFactory] started
13:19:15,468 INFO  [QueueService] Queue[/queue/DLQ] started, fullSize=200000, pageSize=2000, d
ownCacheSize=2000
13:19:15,474 INFO  [QueueService] Queue[/queue/ExpiryQueue] started, fullSize=200000, pageSize
=2000, downCacheSize=2000
13:19:15,476 INFO  [TopicService] Topic[/topic/testTopic] started, fullSize=200000, pageSize=2
000, downCacheSize=2000
13:19:15,478 INFO  [TopicService] Topic[/topic/securedTopic] started, fullSize=200000, pageSiz
e=2000, downCacheSize=2000
13:19:15,479 INFO  [TopicService] Topic[/topic/testDurableTopic] started, fullSize=200000, pag
eSize=2000, downCacheSize=2000
13:19:15,482 INFO  [QueueService] Queue[/queue/testQueue] started, fullSize=200000, pageSize=2
000, downCacheSize=2000
13:19:15,483 INFO  [QueueService] Queue[/queue/A] started, fullSize=200000, pageSize=2000, dow
nCacheSize=2000
13:19:15,485 INFO  [QueueService] Queue[/queue/B] started, fullSize=200000, pageSize=2000, dow
nCacheSize=2000
13:19:15,487 INFO  [QueueService] Queue[/queue/C] started, fullSize=200000, pageSize=2000, dow
nCacheSize=2000
13:19:15,489 INFO  [QueueService] Queue[/queue/D] started, fullSize=200000, pageSize=2000, dow
nCacheSize=2000
13:19:15,490 INFO  [QueueService] Queue[/queue/ex] started, fullSize=200000, pageSize=2000, do
wnCacheSize=2000
13:19:15,501 INFO  [QueueService] Queue[/queue/PrivateDLQ] started, fullSize=200000, pageSize=
2000, downCacheSize=2000
13:19:15,503 INFO  [QueueService] Queue[/queue/PrivateExpiryQueue] started, fullSize=200000, p
ageSize=2000, downCacheSize=2000
13:19:15,507 INFO  [QueueService] Queue[/queue/QueueWithOwnDLQAndExpiryQueue] started, fullSiz
e=200000, pageSize=2000, downCacheSize=2000
13:19:15,508 INFO  [TopicService] Topic[/topic/TopicWithOwnDLQAndExpiryQueue] started, fullSiz
e=200000, pageSize=2000, downCacheSize=2000
13:19:15,511 INFO  [QueueService] Queue[/queue/QueueWithOwnRedeliveryDelay] started, fullSize=
200000, pageSize=2000, downCacheSize=2000
13:19:15,512 INFO  [TopicService] Topic[/topic/TopicWithOwnRedeliveryDelay] started, fullSize=
200000, pageSize=2000, downCacheSize=2000
13:19:15,514 INFO  [QueueService] Queue[/queue/testDistributedQueue] started, fullSize=200000,
pageSize=2000, downCacheSize=2000
13:19:15,519 INFO  [TopicService] Topic[/topic/testDistributedTopic] started, fullSize=200000,
pageSize=2000, downCacheSize=2000
13:19:15,809 INFO  [ConnectionFactoryBindingService] Bound ConnectionManager 'jboss.jca:servic
e=ConnectionFactoryBinding,name=JmsXA' to JNDI name 'java:JmsXA'
13:19:15,834 INFO  [TomcatDeployer] deploy, ctxPath=/jmx-console, warUrl=.../deploy/jmx-consol
e.war/
13:19:16,322 INFO  [Http11Protocol] Starting Coyote HTTP/1.1 on http-127.0.0.1-8080
13:19:16,342 INFO  [AjpProtocol] Starting Coyote AJP/1.3 on ajp-127.0.0.1-8009
13:19:16,480 INFO  [Server] JBoss (MX MicroKernel) [4.2.0.GA (build: SVNTag=JBoss_4_2_0_GA dat
e=200705111440)] Started in 19s:359ms
```

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 Oracle, Sybase, PostgreSQL, MS SQL or MySQL, otherwise you risk losing your data. See Section 6.2 for details about replacing Hypersonic.

4.3. 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.

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

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

A successfull execution log output looks similar to:

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

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

sanity-check:

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

compile:
    [javac] Compiling 5 source files to /home/tim/dev/jboss-messaging/trunk/docs/examples/comm
on/output/classes
    [javac] Compiling 1 source file to /home/tim/dev/jboss-messaging/trunk/docs/examples/queue
/output/classes
```

```
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 1.4.0.GA (1.4)

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

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

It is recommended to run `all` validation examples available in the `example` directory (`queue`, `topic`, `mdb`, `stateless`, etc.). In Chapter 5, we will have a look at each of those examples.

4.4. Accessing JBoss Messaging from a remote client

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

- **Note**

  JBoss Messaging 1.4.0 requires a patched version of jboss-remoting.jar. The version is JBoss Remoting 2.2.2.SP4-brew. This version is available on JBoss EAP 4.3, so if you're using a different version you should download it. The patched jar can be found here [http://repository.jboss.com/jboss/remoting/2.2.2.SP4-brew/lib/] . Please download it and make sure this jar is on your classpath *before* jbossall-client.jar.

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

  JBoss AOP 1.5.5.GA+

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

  (For AOP, sometimes you have to use a specific JAR according to your JVM of choice. Use the most convenient for you)

- $JBOSS_HOME/server/<SERVER_NAME>/lib/javassist.jar

  Javassist 3.5.0.GA-brew+

  http://repository.jboss.com/javassist/3.5.0.GA-brew/lib/

- $JBOSS_HOME/server/<SERVER_NAME>/lib/trove.jar

  trove 1.0.2-brew

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

- log4j

5

Running the Examples

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

- docs/example/queue

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

- docs/example/topic

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

- docs/example/mdb

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

- docs/example/ejb3mdb

  This example demonstrates usage of an EJB3 MDB with JBoss Messaging

- docs/example/stateless

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

- docs/example/mdb-failure

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

- docs/example/secure-socket

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

- docs/example/http

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

- docs/example/web-service

  This example demonstrates JBoss web-service interacting with JBoss Messaging

- docs/example/distributed-queue

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

JBoss AS instances to be running

- docs/example/distributed-topic

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

- docs/example/stateless-clustered

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

- docs/example/bridge

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

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

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

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

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

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

6

Configuration

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

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

The JBoss Messaging service configuration is spread among several configuration files. Depending on the functionality provided by the services it configures, the configuration data is distributed between `messaging-service.xml`, `remoting-bisocket-service.xml`, `xxx-persistence-service.xml` (where xx is the name of your databse) , `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 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.

6.1. 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">

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

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

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

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

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

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

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

```
        <!-- The JAAS security domain to use for JBoss Messaging -->

    <attribute name="SecurityDomain">java:/jaas/messaging</attribute>

    <!-- The default security configuration to apply to destinations - this can be overridden on a per

    <attribute name="DefaultSecurityConfig">
      <security>
          <role name="guest" read="true" write="true" create="true"/>
      </security>
    </attribute>

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

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

    <!-- The default maximum number of times to attempt delivery of a message before sending to the DLQ
         This can be overridden on a per destinatin basis -->

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

6.1.1. ServerPeer attributes

We now discuss the MBean attributes of the ServerPeer MBean.

6.1.1.1. ServerPeerID

The unique id of the server peer. Every node you deploy MUST have a unique id. This applies whether the different nodes form a cluster, or are only linked via a message bridge. The id must be a valid integer.

6.1.1.2. DefaultQueueJNDIContext

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

6.1.1.3. DefaultTopicJNDIContext

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

6.1.1.4. 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.

6.1.1.5. SecurityDomain

The JAAS security domain to be used by this server peer

6.1.1.6. 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.

6.1.1.7. 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.

6.1.1.8. 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.

6.1.1.9. 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.

6.1.1.10. 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.

6.1.1.11. MessageCounterSamplePeriod

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

The default value is `10000` milliseconds.

6.1.1.12. 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).

6.1.1.13. 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).

6.1.1.14. 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.

6.1.1.15. ClusterPullConnectionFactory

The name of the connection factory to use for pulling messages between nodes.

If you wish to turn off message sucking between queues altogether, but retain failover, then you can ommit this attribute altogether

6.1.1.16. DefaultPreserveOrdering

If true, then strict JMS ordering is preserved in the cluster. See the cluster configurations section for more details. Default is false.

6.1.1.17. RecoverDeliveriesTimeout

When failover occurs, already delivered messages will be kept aside, waiting for clients to reconnect. In the case that clients never reconnect (e.g. the client is dead) then eventually these messages will timeout and be added back to the queue. The value is in ms. The default is 5 mins.

6.1.1.18. SuckerPassword

JBoss Messaging internally makes connections between nodes in order to redistribute messages between clustered destinations. These connections are made with the user name of a special reserved user. The password used by that user is specified by this parameter.

> **Warning**
> This must be specified at install time, or the default password will be used. Any one who then knows the default password will be able to gain access to any destinations on the server. This value MUST be changed at install time.

6.1.1.19. StrictTCK

Set to true if you want strict JMS TCK semantiocs

6.1.1.20. Destinations

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

6.1.1.21. MessageCounters

JBoss Messaging provides a message counter for each queue.

6.1.1.22. MessageCountersStatistics

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

6.1.1.23. SupportsFailover

Set to false to prevent server side failover occurring in a cluster when a node crashes.

6.1.1.24. PersistenceManager

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

6.1.1.25. JMSUserManager

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

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

6.1.1.26.1. 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.

6.1.1.26.2. 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`.

6.1.1.26.3. 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`.

6.1.1.26.4. 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.

6.1.1.26.5. 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`.

6.1.1.26.6. 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`.

6.1.1.26.7. ListMessageCountersHTML

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

6.1.1.26.8. ResetAllMesageCounters

This operation resets all message counters to zero.

6.1.1.26.9. ResetAllMesageCounters

This operation resets all message counter histories to zero.

6.1.1.26.10. EnableMessageCounters

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

6.1.1.26.11. DisableMessageCounters

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

6.1.1.26.12. RetrievePreparedTransactions

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

6.1.1.26.13. 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.

6.2. Changing the Database

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

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

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

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

6.3. 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 post-offices are also cluster aware. In a cluster they will automatically route and pull messages

between them in order to provide fully distributed JMS queues and topics.

The post office configuration is found in the xxx-persistence-service.xml file (where xxx is the name of your database).

Here is an example of a post office configuration:

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

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

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

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

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

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

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

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

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

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

      <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 QUEUE_NAME, COND, SELECTOR, CHANNEL_ID, CLUSTERED, ALL_NODES FROM JBM_POSTOFFICE WHE
      ]]></attribute>

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

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

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

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

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

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

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

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

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

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

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

```
                discard_incompatible_packets="true"
                max_bundle_size="64000"
                max_bundle_timeout="30"
                use_incoming_packet_handler="true"
                use_outgoing_packet_handler="false"
                down_thread="false" up_thread="false"
                enable_bundling="false"
                use_send_queues="false"
                sock_conn_timeout="300"
                skip_suspected_members="true"/>
           <MPING timeout="4000"
                        bind_to_all_interfaces="true"
                        mcast_addr="${jboss.messaging.datachanneludpaddress:228.6.6.6}"
                        mcast_port="${jboss.messaging.datachanneludpport:45567}"
                        ip_ttl="8"
                        num_initial_members="2"
                        num_ping_requests="1"/>
           <MERGE2 max_interval="100000"
                   down_thread="false" up_thread="false" min_interval="20000"/>
           <FD_SOCK down_thread="false" up_thread="false"/>
           <VERIFY_SUSPECT timeout="1500" down_thread="false" up_thread="false"/>
           <pbcast.NAKACK max_xmit_size="60000"
                          use_mcast_xmit="false" gc_lag="0"
                          retransmit_timeout="300,600,1200,2400,4800"
                          down_thread="false" up_thread="false"
                          discard_delivered_msgs="true"/>
           <pbcast.STABLE stability_delay="1000" desired_avg_gossip="50000"
                          down_thread="false" up_thread="false"
                          max_bytes="400000"/>
           <pbcast.GMS print_local_addr="true" join_timeout="3000"
                       down_thread="false" up_thread="false"
                       join_retry_timeout="2000" shun="false"
                       view_bundling="true"/>
        </config>
     </attribute>

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

     <!-- We use udp stack for the control channel -->
     <attribute name="ControlChannelConfig">
        <config>
           <UDP
                mcast_addr="${jboss.messaging.controlchanneludpaddress:228.7.7.7}"
                mcast_port="${jboss.messaging.controlchanneludpport:45568}"
                tos="8"
                ucast_recv_buf_size="20000000"
                ucast_send_buf_size="640000"
                mcast_recv_buf_size="25000000"
                mcast_send_buf_size="640000"
                loopback="false"
                discard_incompatible_packets="true"
                max_bundle_size="64000"
                max_bundle_timeout="30"
                use_incoming_packet_handler="true"
                use_outgoing_packet_handler="false"
                ip_ttl="2"
                down_thread="false" up_thread="false"
                enable_bundling="false"/>
           <PING timeout="2000"
                 down_thread="false" up_thread="false" num_initial_members="3"/>
           <MERGE2 max_interval="100000"
                   down_thread="false" up_thread="false" min_interval="20000"/>
           <FD_SOCK down_thread="false" up_thread="false"/>
           <FD timeout="10000" max_tries="5" down_thread="false" up_thread="false" shun="true"/>
           <VERIFY_SUSPECT timeout="1500" down_thread="false" up_thread="false"/>
           <pbcast.NAKACK max_xmit_size="60000"
```

```
                          use_mcast_xmit="false" gc_lag="0"
                          retransmit_timeout="300,600,1200,2400,4800"
                          down_thread="false" up_thread="false"
                          discard_delivered_msgs="true"/>
            <UNICAST timeout="300,600,1200,2400,3600"
                     down_thread="false" up_thread="false"/>
            <pbcast.STABLE stability_delay="1000" desired_avg_gossip="50000"
                          down_thread="false" up_thread="false"
                          max_bytes="400000"/>
            <pbcast.GMS print_local_addr="true" join_timeout="3000" use_flush="true" flush_timeout="3000'
                        down_thread="false" up_thread="false"
                        join_retry_timeout="2000" shun="false"
                        view_bundling="true"/>
            <FRAG2 frag_size="60000" down_thread="false" up_thread="false"/>
            <pbcast.STATE_TRANSFER down_thread="false" up_thread="false" use_flush="true" flush_timeout='
            <pbcast.FLUSH down_thread="false" up_thread="false" timeout="20000" auto_flush_conf="false"/>
        </config>
      </attribute>

    </mbean>
```

6.3.1. The post office has the following attributes

6.3.1.1. DataSource

The datasource the postoffice should use for persisting its mapping data.

6.3.1.2. 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.

6.3.1.3. 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`

6.3.1.4. PostOfficeName

The name of the post office.

6.3.1.5. NodeIDView

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

6.3.1.6. 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.

6.3.1.7. Clustered

If true the post office will take part in a cluster to form distributed queues and topics. If false then it will not participate in the cluster. If false, then all the cluster related attributes will be ignored.

6.3.1.8. 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.

6.3.1.9. CastTimeout

The maximum time to wait for a reply casting message synchronously.

The default value is `5000` milliseconds.

6.3.1.10. MaxConcurrentReplications

The maximum number of concurrent replication requests to make before blocking for replies to come back. This prevents us overwhelming JGroups. This is rarely a good reason to change this.

The default value is `50`

6.3.1.11. ControlChannelConfig

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

The control channel is used for sending request/receiving 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.

6.3.1.12. DataChannelConfig

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

The data channel is used for sending sending/receiving messages from other nodes in the cluster and for replicating session data.

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.

6.4. 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 xxx corresponds to the database name. By default, Messaging ships with a `hsqldb-persistence-service.xml`, which configures the Messaging server to use the in-VM Hypersonic database instance that comes by default with any JBossAS instance.

Warning

The default Persistence Manager configuration is works out of the box with Hypersonic, however it must be stressed that Hypersonic should not be used in a production environment mainly due to its limited support for transaction isolation and its propensity to behave erratically under high load.

The Critique of Hypersonic [http://wiki.jboss.org/wiki/Wiki.jsp?page=ConfigJBossMQDB] wiki page outlines some of the well-known issues occuring when using this database.

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

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

The default Hypersonic persistence configuration file is listed below:

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

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

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

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

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

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

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

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

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

<attribute name="SqlProperties"><![CDATA[
```

```
   CREATE_DUAL=CREATE TABLE JBM_DUAL (DUMMY INTEGER, PRIMARY KEY (DUMMY)) ENGINE = INNODB
   CREATE_MESSAGE_REFERENCE=CREATE TABLE JBM_MSG_REF (CHANNEL_ID BIGINT, MESSAGE_ID BIGINT, TRANSACTION_I
   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_DE
   CREATE_MESSAGE=CREATE TABLE JBM_MSG (MESSAGE_ID BIGINT, RELIABLE CHAR(1), EXPIRATION BIGINT, TIMESTAMP
   CREATE_IDX_MESSAGE_TIMESTAMP=CREATE INDEX JBM_MSG_REF_TIMESTAMP ON JBM_MSG (TIMESTAMP)
   CREATE_TRANSACTION=CREATE TABLE JBM_TX (NODE_ID INTEGER, TRANSACTION_ID BIGINT, BRANCH_QUAL VARBINARY(
   CREATE_COUNTER=CREATE TABLE JBM_COUNTER (NAME VARCHAR(255), NEXT_ID BIGINT, PRIMARY KEY(NAME)) ENGINE
   INSERT_DUAL=INSERT INTO JBM_DUAL VALUES (1)
   CHECK_DUAL=SELECT 1 FROM JBM_DUAL
   INSERT_MESSAGE_REF=INSERT INTO JBM_MSG_REF (CHANNEL_ID, MESSAGE_ID, TRANSACTION_ID, STATE, ORD, PAGE_O
   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 CHANNEL_I
   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=? AND
   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=? A
   LOAD_PAGED_REFS=SELECT MESSAGE_ID, DELIVERY_COUNT, PAGE_ORD, SCHED_DELIVERY FROM JBM_MSG_REF WHERE CHA
   LOAD_UNPAGED_REFS=SELECT MESSAGE_ID, DELIVERY_COUNT, SCHED_DELIVERY FROM JBM_MSG_REF WHERE STATE = 'C'
   LOAD_REFS=SELECT MESSAGE_ID, DELIVERY_COUNT, SCHED_DELIVERY FROM JBM_MSG_REF WHERE STATE = 'C' AND CHA
   UPDATE_REFS_NOT_PAGED=UPDATE JBM_MSG_REF SET PAGE_ORD = NULL WHERE PAGE_ORD BETWEEN ? AND ? AND CHANNE
   SELECT_MIN_MAX_PAGE_ORD=SELECT MIN(PAGE_ORD), MAX(PAGE_ORD) FROM JBM_MSG_REF WHERE CHANNEL_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 FRO
   INSERT_MESSAGE=INSERT INTO JBM_MSG (MESSAGE_ID, RELIABLE, EXPIRATION, TIMESTAMP, PRIORITY, TYPE, HEADE
   INSERT_MESSAGE_CONDITIONAL=INSERT INTO JBM_MSG (MESSAGE_ID, RELIABLE, EXPIRATION, TIMESTAMP, PRIORITY,
   UPDATE_MESSAGE_4CONDITIONAL=UPDATE JBM_MSG SET HEADERS=?, PAYLOAD=? WHERE MESSAGE_ID=?
   INSERT_MESSAGE_CONDITIONAL_FULL=INSERT INTO JBM_MSG (MESSAGE_ID, RELIABLE, EXPIRATION, TIMESTAMP, PRIO
   MESSAGE_ID_COLUMN=MESSAGE_ID
   DELETE_MESSAGE=DELETE FROM JBM_MSG WHERE MESSAGE_ID = ? AND NOT EXISTS (SELECT * FROM JBM_MSG_REF WHER
   INSERT_TRANSACTION=INSERT INTO JBM_TX (NODE_ID, TRANSACTION_ID, BRANCH_QUAL, FORMAT_ID, GLOBAL_TXID) V
   DELETE_TRANSACTION=DELETE FROM JBM_TX WHERE NODE_ID = ? AND TRANSACTION_ID = ?
   SELECT_PREPARED_TRANSACTIONS=SELECT TRANSACTION_ID, BRANCH_QUAL, FORMAT_ID, GLOBAL_TXID FROM JBM_TX WH
   SELECT_MESSAGE_ID_FOR_REF=SELECT MESSAGE_ID, CHANNEL_ID FROM JBM_MSG_REF WHERE TRANSACTION_ID = ? AND
   SELECT_MESSAGE_ID_FOR_ACK=SELECT MESSAGE_ID, CHANNEL_ID FROM JBM_MSG_REF WHERE TRANSACTION_ID = ? AND
   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
   UPDATE_TX=UPDATE JBM_TX SET NODE_ID=? WHERE NODE_ID=?
      ]]></attribute>

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

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

   </mbean>
```

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

6.4.1.1. 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`

6.4.1.2. 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`

6.4.1.3. 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`

6.4.1.4. 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`

6.4.1.5. SupportsBlobOnSelect

Oracle (and possibly other databases) is known to not allow BLOBs to be inserted using a INSERT INTO ... SELECT FROM statement, and requires a two stage conditional insert of messages. If this value is false then such a two stage insert will be used.

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

6.4.1.6. 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.

6.4.1.7. 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`

6.5. 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, CLIENTID VARCHAR(128),
            PRIMARY KEY(USER_ID)) ENGINE = INNODB
            CREATE_ROLE_TABLE=CREATE TABLE JBM_ROLE (ROLE_ID VARCHAR(32) NOT NULL,
            USER_ID VARCHAR(32) NOT NULL, PRIMARY KEY(USER_ID, ROLE_ID))
            ENGINE = INNODB
            SELECT_PRECONF_CLIENTID=SELECT CLIENTID FROM JBM_USER WHERE USER_ID=?
            POPULATE.TABLES.1=INSERT INTO JBM_USER (USER_ID,PASSWD,CLIENTID)
            VALUES ('dilbert','dogbert','dilbert-id')
   ]]></attribute>
</mbean>
```

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

6.5.1.1. 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`

6.5.1.2. 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`

6.5.1.3. 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.

6.6. Configuring Destinations

6.6.1. 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
   </attribute>
</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
      </attribute>
   </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>
....
```

6.6.2. Configuring queues

6.6.2.1. We now discuss the attributes of the Queue MBean

6.6.2.1.1. Name

The name of the queue

6.6.2.1.2. JNDIName

The JNDI name where the queue is bound

6.6.2.1.3. DLQ

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

6.6.2.1.4. ExpiryQueue

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

6.6.2.1.5. RedeliveryDelay

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

6.6.2.1.6. MaxDeliveryAttempts

The maximum number of times delivery of a message will be attempted before sending the message to the DLQ, if configured. If set to -1 (the default), the value from the ServerPeer config is used. Any other setting overrides the value set on the ServerPeer config.

6.6.2.1.7. 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.

6.6.2.1.8. 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.

6.6.2.1.9. CreatedProgrammatically

Returns `true` if the queue was created programmatically

6.6.2.1.10. MessageCount

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

6.6.2.1.11. ScheduledMessageCount

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

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

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

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

Message msg = sess.createMessage();

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

prod.send(msg);
```

6.6.2.1.12. 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.

6.6.2.1.13. Clustered

Clustered destinations must have this set to `true`.

6.6.2.1.14. MessageCounter

Each queue maintains a message counter.

6.6.2.1.15. MessageCounterStatistics

The statistics for the message counter

6.6.2.1.16. MessageCounterHistoryDayLimit

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

6.6.2.1.17. ConsumerCount

The number of consumers currently consuming from the queue.

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

6.6.2.2.1. RemoveAllMessages

Remove (and delete) all messages from the queue.

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

.

6.6.2.2.2. 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

6.6.2.2.3. 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

6.6.2.2.4. 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

6.6.2.2.5. ResetMessageCounter

Resets the message counter to zero.

6.6.2.2.6. ResetMessageCounterHistory

Resets the message counter history.

6.6.2.2.7. ListMessageCounterAsHTML

Lists the message counter in an easy to display HTML format

6.6.2.2.8. ListMessageCounterHistoryAsHTML

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

6.6.3. Configuring topics

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

6.6.3.1.1. Name

The name of the topic

6.6.3.1.2. JNDIName

The JNDI name where the topic is bound

6.6.3.1.3. DLQ

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

6.6.3.1.4. ExpiryQueue

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

6.6.3.1.5. RedeliveryDelay

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

6.6.3.1.6. MaxDeliveryAttempts

The maximum number of times delivery of a message will be attempted before sending the message to the DLQ, if configured. If set to -1 (the default), the value from the ServerPeer config is used. Any other setting overrides the value set on the ServerPeer config.

6.6.3.1.7. 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.

6.6.3.1.8. 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.

6.6.3.1.9. CreatedProgrammatically

Returns `true` if the topic was created programmatically

6.6.3.1.10. 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.

6.6.3.1.11. Clustered

Clustered destinations must have this set to `true`

6.6.3.1.12. MessageCounterHistoryDayLimit

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

6.6.3.1.13. MessageCounters

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

6.6.3.1.14. AllMessageCount

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

6.6.3.1.15. DurableMessageCount

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

6.6.3.1.16. NonDurableMessageCount

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

6.6.3.1.17. AllSubscriptionsCount

The count of all subscriptions on this topic

6.6.3.1.18. DurableSubscriptionsCount

The count of all durable subscriptions on this topic

6.6.3.1.19. NonDurableSubscriptionsCount

The count of all non durable subscriptions on this topic

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

6.6.3.2.1. 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

6.6.3.2.2. ListAllSubscriptions

List all subscriptions of this topic

6.6.3.2.3. ListDurableSubscriptions

List all durable subscriptions of this topic

6.6.3.2.4. ListNonDurableSubscriptions

List all non durable subscriptions of this topic

6.6.3.2.5. ListAllSubscriptionsAsHTML

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

6.6.3.2.6. ListDurableSubscriptionsAsHTML

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

6.6.3.2.7. ListNonDurableSubscriptionsAsHTML

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

6.6.3.2.8. ListAllMessages

Lists all messages for the specified subscription.

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

6.6.3.2.9. ListNonDurableMessages

Lists all non durable messages for the specified subscription.

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

6.6.3.2.10. ListDurableMessages

Lists all durable messages for the specified subscription.

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

6.7. Configuring Connection Factories

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

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

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

You may want to configure additional connection factories, for instance if you want to provide a default client id for a connection factory, or if you want to bind it in different places in JNDI, if you want different connection factories to use different transports, or if you want to selective enable or disable load-balancing and/or automatic

failover for a particular connection factory. Deploying a new connection factory is equivalent with adding a new ConnectionFactory MBean configuration to `connection-factories-service.xml`.

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

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

An example connection factory configuration is presented below:

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

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

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

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

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

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

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

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

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

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

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

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

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

The above example would create a connection factory with pre-configured client ID `myClientID` and bind the connection factory in two places in the JNDI tree: `/MyConnectionFactory` and `/factories/cf`. The connection factory overrides the default values for PreFetchSize, DefaultTempQueueFullSize, DefaultTempQueuePageSize, DefaultTempQueueDownCacheSize and DupsOKBatchSize, SupportsFailover, SupportsLoadBalancing and LoadBalancingFactory. The connection factory will use the default remoting connector. To use a different remoting connector with the connection factory change the `Connector` attribute to specify the service name of the connector you wish

to use.

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

6.7.1.1. ClientID

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

6.7.1.2. JNDIBindings

The list of the JNDI bindings for this connection factory

6.7.1.3. 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.

6.7.1.4. SlowConsumers

If you have very slow consumers, then you probably want to make sure they don't buffer any messages. Since this can prevent them from being consumed by faster consumers.

6.7.1.5. StrictTck

Set this to true if you want strict JMS behaviour as required by the TCK.

6.7.1.6. 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.

6.7.1.7. 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`

6.7.1.8. SupportsLoadBalancing

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

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

node.

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

The default value is `false`

6.7.1.9. SupportsFailover

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

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

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

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

The default value is `false`

6.7.1.10. DisableRemotingChecks

By default, when deploying a connection factory, JBoss Messaging checks that the corresponding JBoss Remoting Connector has "sensible" values. JBoss Messaging is very sensitive to the values and for many of them there's rarely a good reason to change them. To disable such sanity checking set this to false.

> **Warning**
> There is rarely a good reason to disable checking. Only do so if you are absolutely sure in what you are doing

The default value is `false`

6.7.1.11. LoadBalancingFactory

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

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

6.7.1.12. Connector

This specifies which remoting connector this connection factory uses. Different connection factories can use differ-

ent 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.

6.8. 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-bisocket-service.xml

Here is an example bisocket remoting configuration:

```
<config>
   <invoker transport="bisocket">

      <!-- There should be no reason to change these parameters - warning!
           Changing them may stop JBoss Messaging working correctly -->
      <attribute name="marshaller" isParam="true">org.jboss.jms.wireformat.JMSWireFormat</attrib
      <attribute name="unmarshaller" isParam="true">org.jboss.jms.wireformat.JMSWireFormat</attr
      <attribute name="dataType" isParam="true">jms</attribute>
      <attribute name="socket.check_connection" isParam="true">false</attribute>
      <attribute name="timeout" isParam="true">0</attribute>
      <attribute name="serverBindAddress">${jboss.bind.address}</attribute>
      <attribute name="serverBindPort">4457</attribute>
      <attribute name="clientSocketClass" isParam="true">org.jboss.jms.client.remoting.ClientSoc
      <attribute name="serverSocketClass" isParam="true">org.jboss.jms.server.remoting.ServerSoc
      <attribute name="numberOfCallRetries" isParam="true">1</attribute>
      <attribute name="pingFrequency" isParam="true">214748364</attribute>
      <attribute name="pingWindowFactor" isParam="true">10</attribute>
      <attribute name="onewayThreadPool">org.jboss.jms.server.remoting.DirectThreadPool</attribu

      <!-- Periodicity of client pings. Server window by default is twice this figure -->
```

```
               <attribute name="clientLeasePeriod" isParam="true">10000</attribute>

               <!-- Number of seconds to wait for a connection in the client pool to become free -->
               <attribute name="numberOfRetries" isParam="true">10</attribute>

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

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

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

Please note that some of the attributes should not be changed unless you know exactly what you are doing. We will discuss the attributes that you may have a good reason to change:

- clientLeasePeriod - Clients periodically send heartbeats to the server to tell the server they are still alive. If the server does not receive a heartbeat after a certain time it will close down the connection and remove all resources on the server corresponding to the client's session. The clientLeasePeriod determines the period of heartbeats. The server will (by default) close a client if it does not receive a heartbeat in 2 * clientLeasePeriod ms. The actual factor gets automatically resized according to system load. The value is in milliseconds. The defaut value is 10000 ms.
- numberOfRetries - This effectively corresponds to the number of seconds JBoss Remoting will block on the client connection pool waiting for a connection to become free. If you have a very large number of sessions concurrently accessing the server from a client and you are experiencing issues due to not being able to obtain connections from the pool, you may want to consider increasing this value.
- clientMaxPoolSize - JBoss Remoting maintains a client side pool of TCP connections on which to service requests. If you have a very large number of sessions concurrently accessing the server from a client and you are experiencing issues due to not being able to obtain connections from the pool in a timely manner, you may want to consider increasing this value.
- secondaryBindPort - The bisocket transport uses control connections to pass control messages between server and client. If you want to work behind a firewall you may want to specify a particular value for this according to your firewall configuration. This is the address the secondary ServerSocket binds to
- secondaryConnectPort - This is the port the client uses to connect. You may want to specify this to allow clients to work with NAT routers.
- maxPoolSize - This is the number of threads used on the server side to service requests.

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

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

> **Warning**
> There is rarely a good reason to change values in the the bisocket or sslbisocket connector configuration apart from clientLeasePeriod, clientMaxPoolSize, maxRetries, numberOfRetries, secondaryBindPort or secondaryConnectPort. Changing them can cause JBoss Messaging to stop functioning correctly.

6.9. 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-bisocket-service.xml.

If you are using a newer version of JBM in an older version of JBAS then the example bindings in the AS distribution may well be out of date. It is therefore imperative that the relevant sections are overwritten with the remoting configuration from the JBM distribution.

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

7

JBoss Messaging Clustering Notes

7.1. Unique server peer id

JBoss Messaging clustering should work out of the box in most cases with no configuration changes. It is however crucial that every node is assigned a unique server id, as specified in the installation guide.

Every node deployed must have a unique id, including those in a particular LAN cluster, and also those only linked by mesage bridges.

7.2. Clustered destinations

JBoss Messaging clusters JMS queues and topics transparently across the cluster. Messages sent to a distributed queue or topic on one node are consumable on other nodes. To designate that a particular destination is clustered simply set the clustered attribute in the destination deployment descriptor to true.

JBoss Messaging balances messages between nodes, catering for faster or slower consumers to efficiently balance processing load across the cluster.

If you do not want message redistribution between nodes, but still want to retain the other charactereristics of clustered destinations. You can do this by not specifying the attribute ClusterPullConnectionFactoryName on the server peer

7.3. Clustered durable subs

JBoss Messaging durable subscriptions can also be clustered. This means multiple subscribers can consume from the same durable subscription from different nodes of the cluster. A durable subscription will be clustered if it's topic is clustered

7.4. Clustered temporary destinations

JBoss Messaging also supports clustered temporary topics and queues. All temporary topics and queues will be clustered if the post office is clustered

7.5. Non clustered servers

If you don't want your nodes to participate in a cluster, or only have one non clustered server you can set the

clustered attribute on the postoffice to false

7.6. Message ordering in the cluster

If you wish to apply strict JMS ordering to messages, such that a particular JMS consumer consumes messages in the same order as they were produced by a particular producer, you can set the DefaultPreserveOrdering attribute in the server peer to true. By default this is false. The sideeffect of setting this to true is that messages cannot be distributed as freely around the cluster

7.7. Idempotent operations

If the call to send a persistent message to a persistent destination returns successfully with no exception, then you can be sure that the message was persisted. However if the call doesn't return successfully e.g. if an exception is thrown, then you *can't be sure the message wasn't persisted*. Since the failure might have occurred after persisting the message but before writing the response to the caller. This is a common attribute of any RPC type call: You can't tell by the call not returning that the call didn't actually succeed. Whether it's a web services call, an HTTP get request, an ejb invocation the same applies. The trick is to code your application so your operations are *idempotent* i.e. they can be repeated without getting the system into an inconsistent state. With a message system you can do this on the application level, by checking for duplicate messages, and discarding them if they arrive. Duplicate checking is a very powerful technique that can remove the need for XA transactions in many cases.

7.8. Clustered connection factories

If the supportsLoadBalancing attribute of the connection factory is set to true then consecutive create connection attempts will round robin between available servers. The first node to try is chosen randomly

If the supportsFailover attribute of the connection factory is set to true then automatic failover is enabled. This will automatically failover from one server to another, transparently to the user, in case of failure.

If automatic failover is not required or you wish to do manual failover (JBoss MQ style) this can be set to false, and you can supply a standard JMS ExceptionListener on the connection which will be called in case of connection failure. You would then need to manually close the connection, lookup a new connection factory from HA JNDI and recreate the connection.

8

JBoss Messaging XA Recovery Configuration

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

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

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

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

```
 <properties depends="arjuna" name="jta">
    <!--
    Support subtransactions in the JTA layer?
    Default is NO.
    -->
    <property name="com.arjuna.ats.jta.supportSubtransactions" value="NO"/>
    <property name="com.arjuna.ats.jta.jtaTMImplementation"
       value="com.arjuna.ats.internal.jta.transaction.arjunacore.TransactionManagerImple"/>
    <property name="com.arjuna.ats.jta.jtaUTImplementation"
       value="com.arjuna.ats.internal.jta.transaction.arjunacore.UserTransactionImple"/>
    <!--
       *** Add this line to enable recovery for JMS resources using DefaultJMSProvider ***
    -->
    <property name="com.arjuna.ats.jta.recovery.XAResourceRecovery.JBMESSAGING1"
       value="org.jboss.jms.server.recovery.MessagingXAResourceRecovery;java:/DefaultJMSProvider"/>

</properties>
```

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

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

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

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

Please note that to configure the recovery manager to recovery transactions from any node of the cluster it will be necessary to specify a line in the configuration for every node of the cluster

9

JBoss Messaging Message Bridge Configuration

9.1. Message bridge overview

JBoss Messaging includes a fully functional message bridge.

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

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

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

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

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

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

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

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

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

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

- QOS_AT_MOST_ONCE

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

- QOS_DUPLICATES_OK

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

- QOS_ONCE_AND_ONLY_ONCE

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

  Note

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

9.2. Bridge deployment

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

9.3. Bridge configuration

In this section we describe how to configure the message bridge

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

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

   <!-- The JMS provider loader that is used to lookup the source destination -->
```

```
      <depends optional-attribute-name="SourceProviderLoader">
          jboss.messaging:service=JMSProviderLoader,name=JMSProvider</depends>

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

      <!-- The JNDI lookup for the source destination -->
      <attribute name="SourceDestinationLookup">/queue/A</attribute>

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

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

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

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

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

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

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

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

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

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

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

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

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

      <!-- If true then the message id of the message before bridging will be added
          as a header to the message so it is available to the receiver. Can then be
```

```
        sent as correlation id to correlate in a distributed request-response -->
    <attribute name="AddMessageIDInHeader">false</attribute>

  </mbean>
```

We will now discuss each attribute

9.3.1. SourceProviderLoader

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

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

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

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

9.3.2. TargetProviderLoader

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

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

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

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

9.3.3. SourceDestinationLookup

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

An example would be /queue/mySourceQueue

9.3.4. TargetDestinationLookup

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

An example would be /topic/myTargetTopic

9.3.5. SourceUsername

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

9.3.6. SourcePassword

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

9.3.7. TargetUsername

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

9.3.8. TargetPassword

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

9.3.9. QualityOfServiceMode

This integer represents the desired quality of service mode

Possible values are:

- QOS_AT_MOST_ONCE = 0
- QOS_DUPLICATES_OK = 1
- QOS_ONCE_AND_ONLY_ONCE = 2

Please see Section 9.1 for an explanation of what these mean.

9.3.10. Selector

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

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

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

9.3.11. MaxBatchSize

This attribute specifies the maximum number of messages to consume from the source destination before sending

them in a batch to the target destination. It's value must >= 1

9.3.12. MaxBatchTime

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

9.3.13. SubName

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

9.3.14. ClientID

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

9.3.15. FailureRetryInterval

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

9.3.16. MaxRetries

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

9.3.17. AddMessageIDInHeader

If true, then the original message's message id will appended in the message sent to the destination in the header JBossMessage.JBOSS_MESSAGING_BRIDGE_MESSAGE_ID_LIST. If the message is bridged more than once each message-id will be appended. This enables a distributed request-response pattern to be used