

JBoss Messaging User Guide

4.3

JBoss Enterprise Application Platform



ISBN: N/A

Publication date: Oct, 2007

This User Guide documents relevant information regarding the usage of JBoss Messaging 1.4.0 for JBoss Enterprise Application Platform 4.3

JBoss Messaging User Guide: JBoss Enterprise Application Platform

Copyright © 2008 Red Hat, Inc

Copyright © 2008 Red Hat, Inc. This material may only be distributed subject to the terms and conditions set forth in the Creative Commons Attribution-Noncommercial-Share Alike 3.0 Unported License (which is presently available at <http://creativecommons.org/licenses/by-nc-sa/3.0/>).

Red Hat and the Red Hat "Shadow Man" logo are registered trademarks of Red Hat, Inc. in the United States and other countries.

All other trademarks referenced herein are the property of their respective owners.

The GPG fingerprint of the security@redhat.com key is:

CA 20 86 86 2B D6 9D FC 65 F6 EC C4 21 91 80 CD DB 42 A6 0E

1801 Varsity Drive
Raleigh, NC 27606-2072
USA
Phone: +1 919 754 3700
Phone: 888 733 4281
Fax: +1 919 754 3701
PO Box 13588
Research Triangle Park, NC 27709
USA

1. About this book	1
2. Introduction	3
1. Feedback	3
2. Other Manuals	3
3. JBoss Messaging - A Quick Tour	5
1. Limitations of JBossMQ	5
2. JBoss Messaging Features	5
3. Clustering Features	6
4. Compatibility with JBossMQ	7
4. Running the Examples	9
5. Configuration	11
1. Configuring the ServerPeer	11
1.1. ServerPeer attributes	14
2. Changing the Database	20
3. Configuring the Post office	20
3.1. The post office has the following attributes	23
4. Configuring the Persistence Manager	25
4.1. We now discuss the MBean attributes of the PersistenceManager MBean	27
5. Configuring the JMS user manager	28
5.1. We now discuss the MBean attributes of the JMSUserManager MBean	29
6. Configuring Destinations	30
6.1. Pre-configured destinations	30
6.2. Configuring queues	33
6.3. Configuring topics	37
7. Configuring Connection Factories	41
7.1. We now discuss the MBean attributes of the ConnectionFactory MBean	42
8. Configuring the remoting connector	44
9. ServiceBindingManager	45
10. Configuring the callback	45
11. Accessing JBoss Messaging from a remote client	46
6. JBoss Messaging Clustering Configuration	47
7. JBoss Messaging XA Recovery Configuration	51
8. JBoss Messaging Message Bridge Configuration	53
1. Message bridge overview	53
2. Bridge deployment	54
3. Bridge configuration	54
3.1. SourceProviderLoader	56
3.2. TargetProviderLoader	57
3.3. SourceDestinationLookup	57
3.4. TargetDestinationLookup	57
3.5. SourceUsername	57
3.6. SourcePassword	57
3.7. TargetUsername	57
3.8. TargetPassword	58
3.9. QualityOfServiceMode	58

3.10. Selector	58
3.11. MaxBatchSize	58
3.12. MaxBatchTime	58
3.13. SubName	58
3.14. ClientID	59
3.15. FailureRetryInterval	59
3.16. MaxRetries	59
3.17. AddMessageIDInHeader	59

About this book

The goal of this book is to give you an overview of JBoss Messaging, and demonstrate some of its features and capability to provide a high-performance and robust messaging system for Enterprise Java Applications.

JBoss Messaging is the new state-of-the-art enterprise messaging system from JBoss, providing superior performance, reliability and scalability with high throughput and low latency. JBoss Messaging has replaced JBossMQ as the default JMS provider in JBoss Enterprise Application Platform 4.3. The version used is JBoss Messaging 1.4.0.GA. You should use this version or later with the examples demonstrating JBoss Messaging. The book is not intended to teach you the basics of Enterprise Messaging. Use it to familiarize with JBoss Messaging features and various configurations.

Contributors.

Tim Fox, Clebert Suconic, Andy Taylor, Ovidiu Feodorov, Vinu S Renish, Sergey Koshcheyev, Ron Sigal, Madhu Konda, Jay Howell, Tyronne Wickramaratne, Aaron Walker, Adrian Brock, Rajdeep Dua, Tom Elrod, Alex Fu, Juha Lindfors, Alexey Loubyansky, Luc Texier, Scott Stark, Jay Howell, David Boeren, Mike Clark, Tyronne Wickramaratne, Mark Little, Pete Bennett.

Introduction

JBoss Messaging provides an open source and standards-based messaging platform and is an integral part of Red Hat's strategy for enterprise messaging. It is a complete rewrite of JBossMQ, the legacy JBoss JMS provider, and offers improved performance in both single node and clustered environments. It also features a much better modular architecture that allows easy adding of new features in the future.



Note

JBoss Messaging 1.4.0.GA is the default JMS provider in JBoss Enterprise Application Platform 4.3.

1. Feedback

If you spot a typo in this guide, or if you have thought of a way to make this manual better, we would love to hear from you! Submit a report in [JIRA](http://jira.jboss.com/jira/browse/JBPAPP) [http://jira.jboss.com/jira/browse/JBPAPP] against the Product: JBoss Enterprise Application Platform, Version: *<version>*, Component: *Doc*. If you have a suggestion for improving the documentation, try to be as specific as possible. If you have found an error, include the section number and some of the surrounding text so we can find it easily.

2. Other Manuals

If you are looking for detailed product information refer to the manuals available online at <http://www.redhat.com/docs/manuals/jboss>.

JBoss Messaging - A Quick Tour

1. Limitations of JBossMQ

JBossMQ has two fundamental limitations:

- JBossMQ is based on SpyderMQ (the open source project) which is a non-clustered broker.
- The threading model and the overall design of the non-clustered broker leads to performance limitations in certain high load usage scenarios.

2. JBoss Messaging Features

JBoss Messaging implements a high-performance and robust messaging core that is designed to support the largest and most heavily utilized Service Oriented Architectures(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 node's CPU cycles. It comes with no single point of failure and no single point of bottleneck, sophisticated and fully configurable message expiration handling and XA transaction recovery. Thus providing a highly scalable and performant clustering implementation. It 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.



Note

JMS compliance: A fully compatible and Sun certified JMS 1.1 implementation, that currently works with JBoss Enterprise Application Platform or JBoss Application Server version 4.2 or later.

JBoss Messaging contains a host of other features, including:

- 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

3. 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.

4. Compatibility with JBossMQ

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 JBossMQ 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.

**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.

Running the Examples

In the directory `docs/examples`, you will find a set of examples demonstrating JBoss Messaging working in various examples. To run the example, navigate to the folder in a command line prompt, and type `ant`. Apache Ant and your JBoss installation path must be present in your environment `path` variable, in order to be able to run the example. Make sure you start the JBoss server before trying to run these examples.

- `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 occurring 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 interacts 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

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.

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`, `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.

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 an integer -->

  <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 destination basis -->

    <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 destination basis -->

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

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

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

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

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

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

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

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

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

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

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

    <!-- The maximum amount of time for a client to wait for failover to
```

```

complete
    on the server side after it has detected failure -->

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

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

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

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

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

    <!-- Use XA when pulling persistent messages from a remote node to
this one. -->

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

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

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

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

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

    <!-- 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
    </depends>

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

<depends>jboss.messaging:service=Connector,transport=bisocket</depends>

</mbean>

```



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

1.1. ServerPeer attributes

We now discuss the MBean attributes of the ServerPeer MBean.

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.

1.1.2. DefaultQueueJNDIContext

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

1.1.3. DefaultTopicJNDIContext

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

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.

1.1.5. SecurityDomain

The JAAS security domain to be used by this server peer

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 messages 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.

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.

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.

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.

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.

1.1.11. MessageCounterSamplePeriod

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

The default value is 10000 milliseconds.

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).

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).

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 maximum number of days for which to store message counter history. It can be overridden on a per destination basis.

1.1.15. ClusterPullConnectionFactory

The name of the connection factory to use for pulling messages between nodes. You will not normally need to change this.

1.1.16. UseXAForMessagePull

If true, then move a reliable message from one node to another in an XA transaction. Relaxing this gives better performance at the expense of some reliability. See the cluster configurations section for more details. Default is true.

1.1.17. DefaultPreserveOrdering

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

1.1.18. 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.

1.1.19. 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.

1.1.20. Destinations

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

1.1.21. MessageCounters

JBoss Messaging provides a message counter for each queue.

1.1.22. MessageCountersStatistics

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

1.1.23. SupportsFailover

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

1.1.24. PersistenceManager

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

1.1.25. JMSUserManager

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

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

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.

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`.

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`.

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.

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`.

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`.

1.1.26.7. ListMessageCountersHTML

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

1.1.26.8. ResetAllMessageCounters

This operation resets all message counters to zero.

1.1.26.9. ResetAllMessageCounters

This operation resets all message counter histories to zero.

1.1.26.10. EnableMessageCounters

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

1.1.26.11. DisableMessageCounters

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

1.1.26.12. RetrievePreparedTransactions

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

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.

2. Changing the Database

The JMS service in the JBoss AS uses relational databases to persist its messages. For improved performance, you should change the JMS service to take advantage of the external database. To do that, you need to replace the file

`jboss-as/server/production/deploy/jboss-messaging.sar/clustered-hsqldb-persistence-service.xml` with a file in `jboss-as/docs/examples/jms/` depending on your external database and restart your server.

- MySQL: `mysql-persistence-service.xml`
- PostgreSQL: `postgresql-persistence-service.xml`
- Oracle: `oracle-persistence-service.xml`
- Sybase: `sybase-persistence-service.xml`
- MS SQL Server: `mssql-persistence-service.xml`

For the default and all configurations, replace the files

`jboss-as/server/default/deploy/jboss-messaging.sar/hsqldb-persistence-service.xml` and

`jboss-as/server/all/deploy/jboss-messaging.sar/clustered-hsqldb-persistence-service.xml` respectively.

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.

You can configure a JCA datasource using an example from `jboss-as/docs/examples/jca` and copying to `jboss-as/server/<config-name>/deploy`. By default JBoss Messaging uses `DefaultDS`.

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 represents 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_NAME VARCHAR(255), COND VARCHAR(1023),
      SELECTOR VARCHAR(1023), CHANNEL_ID BIGINT, CLUSTERED CHAR(1),
      ALL_NODES CHAR(1), PRIMARY KEY(POSTOFFICE_NAME, NODE_ID,
    QUEUE_NAME))
    ENGINE = INNODB
    INSERT_BINDING=INSERT INTO JBM_POSTOFFICE (POSTOFFICE_NAME,
    NODE_ID, QUEUE_NAME,
    COND, SELECTOR, CHANNEL_ID, CLUSTERED, ALL_NODES)
    VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)
  ]]></attribute>
```

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

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

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

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

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

    <attribute name="GroupName">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>

    <!-- Enable this when the JGroups multiplexer comes of age
    <attribute
name="ChannelFactoryName">jgroups.mux:name=Multiplexer</attribute>
    <attribute name="ControlChannelName">udp-sync</attribute>
    <attribute name="DataChannelName">udp</attribute>
    <attribute name="ChannelPartitionName">
        ${jboss.partition.name:DefaultPartition}-JMS
    </attribute>
    -->

    <!-- JGroups stack configuration for the data channel - used when
casting
        messages across the cluster -->

    <attribute name="DataChannelConfig">
        <config>
            <UDP mcast_rcv_buf_size="500000"
                down_thread="false"
                ip_mcast="true"
                mcast_send_buf_size="32000"
                mcast_port="45567"
                ucast_rcv_buf_size="500000"
                use_incoming_packet_handler="false"
```

```
        mcast_addr="228.8.8.8"
        use_outgoing_packet_handler="true"
        loopback="true"
        ucast_send_buf_size="32000"
        ip_ttl="32"/>
    <AUTOCONF    down_thread="false"
                up_thread="false"/>
    <PING    timeout="2000"
            down_thread="false"
            num_initial_members="3"
            up_thread="false"/>
    ... (truncated)
</config>
</attribute>

<!-- JGroups stack configuration to use for the control channel -
      used for bind/unbind requests amongst others -->

<attribute name="ControlChannelConfig">
    <config>
        <UDP
            mcast_rcv_buf_size="500000"
            down_thread="false"
            ip_mcast="true"
            mcast_send_buf_size="32000"
            mcast_port="45568"
            ucast_rcv_buf_size="500000"
            use_incoming_packet_handler="false"
            mcast_addr="228.8.8.8"
            use_outgoing_packet_handler="true"
            loopback="true"
            ucast_send_buf_size="32000"
            ip_ttl="32"/>
        <AUTOCONF down_thread="false"
                up_thread="false"/>
        <PING timeout="2000"
            down_thread="false"
            num_initial_members="3"
            up_thread="false"/>
        <MERGE2 max_interval="10000"
            down_thread="false"
            min_interval="5000"
            up_thread="false"/>
        ... (truncated)
    </config>
</attribute>
</mbean>
```

3.1. The post office has the following attributes

3.1.1. DataSource

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

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.

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`

3.1.4. PostOfficeName

The name of the post office.

3.1.5. NodeIDView

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

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.

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.

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.

3.1.9. CastTimeout

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

The default value is 5000 milliseconds.

3.1.10. 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>.

3.1.11. 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.

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>.

4. Configuring the Persistence Manager

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

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

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



Warning

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



Warning

The *Critique of Hypersonic*

[<http://wiki.jboss.org/wiki/Wiki.jsp?page=ConfigJBossMQDB>] wiki page outlines some of the well-known issues occurring 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 supported 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_MESSAGE=CREATE TABLE JBM_MSG (MESSAGE_ID BIGINT, RELIABLE
CHAR(1),
    EXPIRATION BIGINT, TIMESTAMP BIGINT, PRIORITY TINYINT, HEADERS
MEDIUMBLOB,
```



```

        PAYLOAD LONGBLOB, CHANNEL_COUNT INTEGER, TYPE TINYINT, PRIMARY
KEY (MESSAGE_ID))
        ENGINE = INNODB
        CREATE_MESSAGE_REFERENCE=CREATE TABLE JBM_MSG_REF (CHANNEL_ID
BIGINT,
        MESSAGE_ID BIGINT REFERENCES JBM_MSG(MESSAGE_ID), TRANSACTION_ID
BIGINT,
        STATE CHAR(1), ORD BIGINT, PAGE_ORD BIGINT, DELIVERY_COUNT
INTEGER,
        SCHED_DELIVERY BIGINT, PRIMARY KEY(CHANNEL_ID, MESSAGE_ID))
ENGINE = INNODB
        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_DELIVERY)
        CREATE_TRANSACTION=CREATE TABLE JBM_TX (NODE_ID INTEGER,
TRANSACTION_ID BIGINT,
        BRANCH_QUAL VARBINARY(254), FORMAT_ID INTEGER, GLOBAL_TXID
VARBINARY(254),
        PRIMARY KEY (TRANSACTION_ID)) ENGINE = INNODB
        CREATE_COUNTER=CREATE TABLE JBM_COUNTER (NAME VARCHAR(255),
NEXT_ID BIGINT,
        PRIMARY KEY(NAME)) ENGINE = INNODB
        INSERT_MESSAGE_REF=INSERT INTO JBM_MSG_REF (CHANNEL_ID,
MESSAGE_ID, TRANSACTION_ID,
        STATE, ORD, PAGE_ORD, DELIVERY_COUNT, SCHED_DELIVERY)
VALUES (?, ?, ?, ?, ?, ?, ?, ?)
        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_ID=? AND STATE='C'
        UPDATE_PAGE_ORDER=UPDATE JBM_MSG_REF SET PAGE_ORD = ? WHERE
MESSAGE_ID=? AND CHANNEL_ID=?
        ... (truncated)
    ]]></attribute>

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

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

```

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

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`

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`

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`

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`

4.1.5. SQLProperties

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

4.1.6. MaxParams

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

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

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>
```

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

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`

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`

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. Configuring Destinations

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 individual 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>
....
```

6.2. Configuring queues

6.2.1. We now discuss the attributes of the Queue MBean

6.2.1.1. Name

The name of the queue

6.2.1.2. JNDIName

The JNDI name where the queue is bound

6.2.1.3. DLQ

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

6.2.1.4. ExpiryQueue

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

6.2.1.5. RedeliveryDelay

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

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.

6.2.1.7. 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 messages 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 specified then the default security configuration from the Server Peer will be used.

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 by 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 subscription 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.

6.2.1.9. CreatedProgrammatically

Returns `true` if the queue was created programmatically

6.2.1.10. MessageCount

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

6.2.1.11. 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);
```

6.2.1.12. 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.

6.2.1.13. Clustered

Clustered destinations must have this set to `true`.

6.2.1.14. MessageCounter

Each queue maintains a message counter.

6.2.1.15. MessageCounterStatistics

The statistics for the message counter

6.2.1.16. MessageCounterHistoryDayLimit

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

6.2.1.17. ConsumerCount

The number of consumers currently consuming from the queue.

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

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

6.2.2.2. 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

6.2.2.3. 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

6.2.2.4. 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

6.2.2.5. ResetMessageCounter

Resets the message counter to zero.

6.2.2.6. ResetMessageCounterHistory

Resets the message counter history.

6.2.2.7. ListMessageCounterAsHTML

Lists the message counter in an easy to display HTML format

6.2.2.8. ListMessageCounterHistoryAsHTML

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

6.3. Configuring topics

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

6.3.1.1. Name

The name of the topic

6.3.1.2. JNDIName

The JNDI name where the topic is bound

6.3.1.3. DLQ

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

6.3.1.4. ExpiryQueue

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

6.3.1.5. RedeliveryDelay

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

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.

6.3.1.7. 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 messages 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 specified then the default security configuration from the Server Peer will be used.

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 by 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 subscription 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.

6.3.1.9. CreatedProgrammatically

Returns `true` if the topic was created programmatically

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.

6.3.1.11. Clustered

Clustered destinations must have this set to `true`

6.3.1.12. MessageCounterHistoryDayLimit

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

6.3.1.13. MessageCounters

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

6.3.1.14. AllMessageCount

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

6.3.1.15. DurableMessageCount

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

6.3.1.16. NonDurableMessageCount

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

6.3.1.17. AllSubscriptionsCount

The count of all subscriptions on this topic

6.3.1.18. DurableSubscriptionsCount

The count of all durable subscriptions on this topic

6.3.1.19. NonDurableSubscriptionsCount

The count of all non durable subscriptions on this topic

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

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

6.3.2.2. ListAllSubscriptions

List all subscriptions of this topic

6.3.2.3. ListDurableSubscriptions

List all durable subscriptions of this topic

6.3.2.4. ListNonDurableSubscriptions

List all non durable subscriptions of this topic

6.3.2.5. ListAllSubscriptionsAsHTML

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

6.3.2.6. ListDurableSubscriptionsAsHTML

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

6.3.2.7. ListNonDurableSubscriptionsAsHTML

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

6.3.2.8. ListAllMessages

Lists all messages for the specified subscription.

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

6.3.2.9. ListNonDurableMessages

Lists all non durable messages for the specified subscription.

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

6.3.2.10. ListDurableMessages

Lists all durable messages for the specified subscription.

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

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.

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

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

7.1.2. JNDIBindings

The list of the JNDI bindings for this connection factory

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.

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.

7.1.5. StrictTck

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

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 omitted.

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

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`

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 `HASNDI` and recreate the connection using that.

The default value is `false`

7.1.10. LoadBalancingFactory

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

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

7.1.11. Connector

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

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

8. 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 protocol. 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
```

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

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

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

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

10. Configuring the callback

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

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

property in your java arguments:

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

The client port will be selected randomly for any non used port. But if you defined `-Djboss.messaging.callback.bind.port=NumericPort` in your System Properties that number is going to be used for the call back client port.

11. Accessing JBoss Messaging from a remote client

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

- `jboss-messaging-client.jar`
- `jbossall-client.jar`
- `jboss-aop.jar`
- `javassist.jar`
- `trove.jar`
- `log4j`

JBoss Messaging Clustering Configuration

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 message bridges.



Note

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.

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.

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

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

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

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 side-effect of setting this to true is that messages cannot be distributed as freely around the cluster

When pulling reliable messages from one node to another, by default JBoss Messaging uses an XA transaction to ensure that message was removed from one node and added to another transactionally. Using XA transactions is a fairly heavyweight operation. If you are willing to relax the reliability guarantee somewhat in order to gain some performance then you can set the

attribute `UseXAForMessagePull` in server peer to false. By default it is true



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.

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

```
<service-config
name="jboss.messaging:service=Connector,transport=bisocket"
delegateClass="org.jboss.services.binding.AttributeMappingDelegate">
  <delegate-config>
    <attribute name="Configuration"><![CDATA[
<config>
  <invoker transport="bisocket">
    <attribute name="marshaller" isParam="true">
      org.jboss.jms.wireformat.JMSWireFormat
    </attribute>
    <attribute name="unmarshaller" isParam="true">
      org.jboss.jms.wireformat.JMSWireFormat
    </attribute>
    <attribute name="dataType" isParam="true">jms</attribute>
    <attribute name="socket.check_connection"
isParam="true">>false</attribute>
    <attribute name="timeout" isParam="true">0</attribute>
    <attribute
name="serverBindAddress">${jboss.bind.address}</attribute>
    <attribute name="serverBindPort">4657</attribute>
    <attribute name="leasePeriod">10000</attribute>
    <attribute name="clientSocketClass" isParam="true">
      org.jboss.jms.client.remoting.ClientSocketWrapper
    </attribute>
    <attribute name="serverSocketClass">
      org.jboss.jms.server.remoting.ServerSocketWrapper
```

```
        </attribute>
        <attribute name="numberOfRetries"
isParam="true">1</attribute>
        <attribute name="numberOfCallRetries"
isParam="true">1</attribute>
        <attribute name="clientMaxPoolSize"
isParam="true">50</attribute>
    </invoker>
    <handlers>
        <handler subsystem="JMS">
            org.jboss.jms.server.remoting.JMSServerInvocationHandler
        </handler>
    </handlers>
</config>
]]></attribute>
</delegate-config>
<binding port="4657"/>
</service-config>
```



Warning

You must ensure that the config (like above) is identical to that in `remoting-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 is out of date and you will need to copy the config from `remoting-bisocket-service.xml`.

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

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

JBoss Messaging Message Bridge Configuration

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 connection 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 available 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.

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

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

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

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 `ha-jndi-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

3.3. SourceDestinationLookup

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

An example would be `/queue/mySourceQueue`

3.4. TargetDestinationLookup

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

An example would be `/topic/myTargetTopic`

3.5. SourceUsername

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

3.6. SourcePassword

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

3.7. TargetUsername

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

connection

3.8. TargetPassword

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

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 1, “Message bridge overview”](#) for an explanation of what these mean.

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>

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

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.

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

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

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

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'

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

