

How to

Use Java Messaging Service with AltioLive

- Tools:** Application Manager, Presentation Server Administration, Altio DB Manager.
- Duration:** 25 minutes
- Skill Level:** Medium
- Summary:** An introduction to using JMS with AltioLive. The example shows how to use the Altio DB Manager with JBoss2.

Integra SP

88 Wood Street London
EC2V 7RS
United Kingdom

www.altio.com
tel: +44 (0) 20 8528 1045

Contents

What is JMS (Java Messaging Service).....	3
Publish/Subscribe Messaging Domain (Topic based)	5
Point-to-Point Messaging Domain (Queue based)	6
Message Consumption	7
JNDI (Java Naming and Directory Interface).....	8
Overview.....	8
JNDI Access via the Global Context.....	8
JNDI Access via the ENC.....	9
JBoss Installation/Configuration Notes	10
Configuring JNDI - Altio.....	12
Enable JBoss connection.....	12
JNDI Settings	14
JNDI Access via the Global Context.....	15
Using JMS with Altio	17

Overview	17
Configuring a Service Function	18
Configuring a Datapool	19
Configuring AltioDB	20
Configuring ROS (JMS) Techlet	23
Configure JBoss	23
Configure Altio	26
Configure AltioDB	26
ENC Configuration	27
Deploying Altio	27

What is JMS (Java Messaging Service)

Messaging enables distributed communication that is *loosely coupled*. A component sends a message to a destination, and the recipient can retrieve the message from the destination. However, the sender and the receiver do not have to be available at the same time in order to communicate. In fact, the sender does not need to know anything about the receiver; nor does the receiver need to know anything about the sender. The sender and the receiver need to know only what message format and what destination to use.

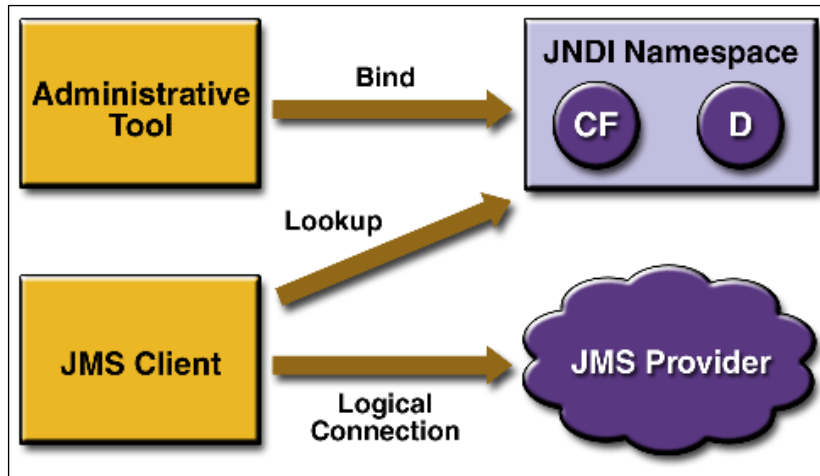
JMS is a Java API that allows applications to create, send, receive, and read messages. It is primarily used for reliable (ensure that a message is delivered once and only once) asynchronous message delivery, in a one-to-one or one-to-many configuration.

A JMS application is composed of the following parts.

- A **JMS provider** is a messaging system that implements the JMS interfaces and provides administrative and control features. An implementation of the J2EE platform at release 1.3 includes a JMS provider.
- **JMS clients** are the programs or components, written in the Java™ programming language, that produce and consume messages.
- **Messages** are the objects that communicate information between JMS clients.
- **Administered objects** are pre-configured JMS objects created by an administrator for the use of clients. The two kinds of administered objects are destinations and connection factories.

- **Native clients** are programs that use a messaging product's native client API instead of the JMS API. An application first created before the JMS API became available and subsequently modified is likely to include both JMS and native clients.

Administrative tools allow you to bind destinations and connection factories into a Java Naming and Directory Interface™ (JNDI) API namespace. A JMS client can then look up the administered objects in the namespace and then establish a logical connection to the same objects through the JMS provider.

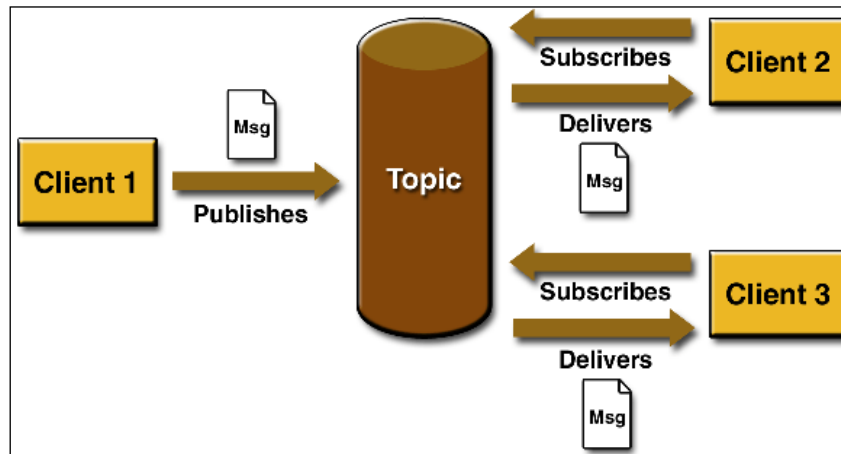


Publish/Subscribe Messaging Domain (Topic based)

In a publish/subscribe (pub/sub) product or application, clients address messages to a topic. Publishers and subscribers are generally anonymous and may dynamically publish or subscribe to the content hierarchy. The system takes care of distributing the messages arriving from a topic's multiple publishers to its **multiple** subscribers. Topics retain messages only as long as it takes to distribute them to current subscribers.

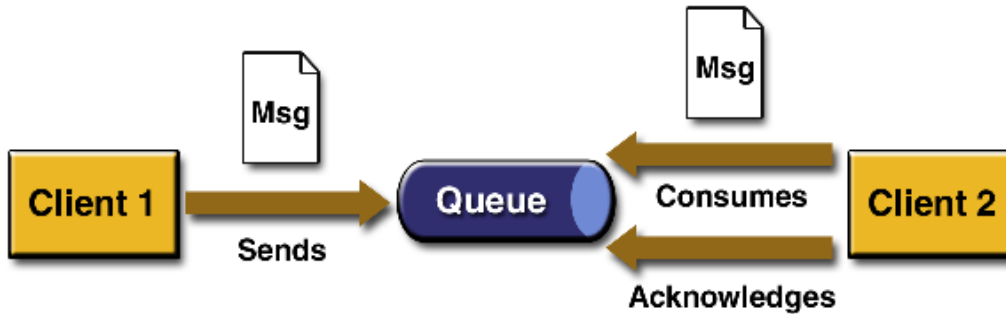
Publishers and subscribers have a timing dependency. A client that subscribes to a topic can consume only messages published after the client has created a subscription, and the subscriber must continue to be active in order for it to consume messages. (In the JMS API, *Durable Subscriptions* can receive messages sent while the subscribers are not active).

Use pub/sub messaging when each message can be processed by zero, one, or many consumers.



Point-to-Point Messaging Domain (Queue based)

A point-to-point (PTP) product or application is built around the concept of message queues, senders, and receivers. Each message is addressed to a specific queue, and receiving clients extract messages from the queue(s) established to hold their messages. Queues retain all messages sent to them until the messages are consumed or until the messages expire.



PTP messaging is used when every message you send must be processed successfully by one consumer. It has the following characteristics:

- Each message has only one consumer.
- A sender and a receiver of a message have no timing dependencies. The receiver can fetch the message whether or not it was running when the client sent the message.
- The receiver acknowledges the successful processing of a message.

Message Consumption

Messaging products are inherently asynchronous in that no fundamental timing dependency exists between the production and the consumption of a message. However, the JMS Specification uses this term in a more precise sense. Messages can be consumed in either of two ways:

- **Synchronously.** A subscriber or a receiver explicitly fetches the message from the destination by calling the receive method. The receive method can block until a message arrives or can time out if a message does not arrive within a specified time limit.
- **Asynchronously.** A client can register a *message listener* with a consumer. A message listener is similar to an event listener. Whenever a message arrives at the destination, the JMS provider delivers the message by calling the listener's `onMessage` method, which acts on the contents of the message.

JNDI (Java Naming and Directory Interface)

Overview

JNDI is a standard API for accessing objects bound within a naming service. A naming service maps string names to objects, and is a useful way of letting code obtain resources without having to know the resource implementation or location. There are two ways to access JNDI in a J2EE setup: via the web applications Environment Naming Context (ENC) and via the global context.

JNDI Access via the Global Context

This is useful if you are using a naming service that is **not** closely tied in with your J2EE server. This is the case when AltioLive is deployed in a separate environment from the J2EE naming service. As JNDI is an interface, you need to tell it information to allow it to find the implementation of that interface as well as how/where to connect to the target naming service. Some of the more common parameters are:

- `java.naming.factory.initial`
- `java.naming.provider.url`
- `java.naming.factory.url.pkgs`

A full list of parameters can be found in the javadocs for the JNDI classes.

JNDI parameters allow customization of JNDI via zero or more name value pairs. These name value pairs map directly to the hashtable that is used to configure the InitialContext access to the naming service (see any JNDI book for details).

Topic Connection Factory: Used with JMS, for publish-and-subscribe messaging. Before using any topics in JMS, you must first create a connection to a topic using a topic connection factory. This object is usually located in the JNDI namespace under the name "TopicConnectionFactory".

Queue Connection Factory: Used with JMS, for point-to-point messaging. In JMS before using any queues, you must first create a connection to a queue using a queue connection factory. This object is normally located in the JNDI namespace under the name "QueueConnectionFactory".

JNDI Access via the ENC

If AltioLive is deployed in the same J2EE environment as the JNDI service, then ENC is used to access the naming service. The information available via the ENC is controlled by the information in the web.xml file (located under WEB-INF in the web application). If the J2EE server you are using has a graphical interface, then this information can usually be set from there. The names used to access retrieve objects from the ENC are normally of the form "java:comp/env/jms/myTopic".

To use the ENC you will need to clear any JNDI parameters.

JBoss Installation/Configuration Notes

1. Unzip JBoss to a directory (e.g. **C:\Program Files\jboss\JBoss-2.4.1_Tomcat-3.2.3**). We are using version 2.4 with Tomcat included. Version 3 (the latest JBoss would require Altio to run with a 1.4 jvm – Studio is supplied with a 1.3 jvm)
2. Start JBoss (**C:\Program Files\jboss\JBoss-2.4.1_Tomcat-3.2.3\jboss\bin\run_with_tomcat.bat**)
3. Launch a browser to access the JBoss console (<http://localhost:8082>). Click on the Admin button to add Topics:

Agent View [JMX RI/1.0]

Filter by object name:

This agent is registered on the domain *DefaultDomain*. Admin

This page contains **66** MBean(s).

4. For the **ROS JMS** demo, add three topics. Domain and Java Class stay the same. Simply modify the name value in Keys to add **newBidService**, **newSaleService**, and **newSaleDatapool**.

Specify the object name and java class of the MBean to add, delete or view the constructors of
(Optionally provide a class loader name for loading the specified class.)

Domain:	<input type="text" value="JBossMQ"/>
Keys:	<input type="text" value="service=Topic,name=newBidService"/>
Java Class:	<input type="text" value="org.jboss.mq.server.TopicManager"/>
Class Loader:	<input type="text"/>

Action:

These topics can also be added directly to the configuration file (**C:\Program Files\jboss\JBoss-2.4.1_Tomcat-3.2.3\jboss\conf\tomcat\jboss.jcml**):

```
<!-- ===== -->
<!-- Add your custom MBeans here -->
<!-- ===== -->
<mbean code="org.jboss.mq.server.TopicManager" name="JBossMQ:service=Topic,name=newSaleService"/>
<mbean code="org.jboss.mq.server.TopicManager" name="JBossMQ:service=Topic,name=newBidService"/>
<mbean code="org.jboss.mq.server.TopicManager" name="JBossMQ:service=Topic,name=newSaleDatapool"/>
```

Configuring JNDI - Altio

Enable JBoss connection

To enable AltioLive to communicate to JBoss, you need to copy the **JBoss client** jars listed below from **C:\Program Files\jboss\JBoss-2.4.1_Tomcat-3.2.3\jboss\client** to AltioLive's **WEB-INF\lib** directory:

...\altiolive_studio54\tomcat\webapps\altio54\WEB-INF\lib)

- **jboss-client.jar**
- **jboss-j2ee.jar**
- **jbossmq-client.jar**
- **jbossx-client.jar**
- **jnp-client.jar**
- **oswego-concurrent.jar**

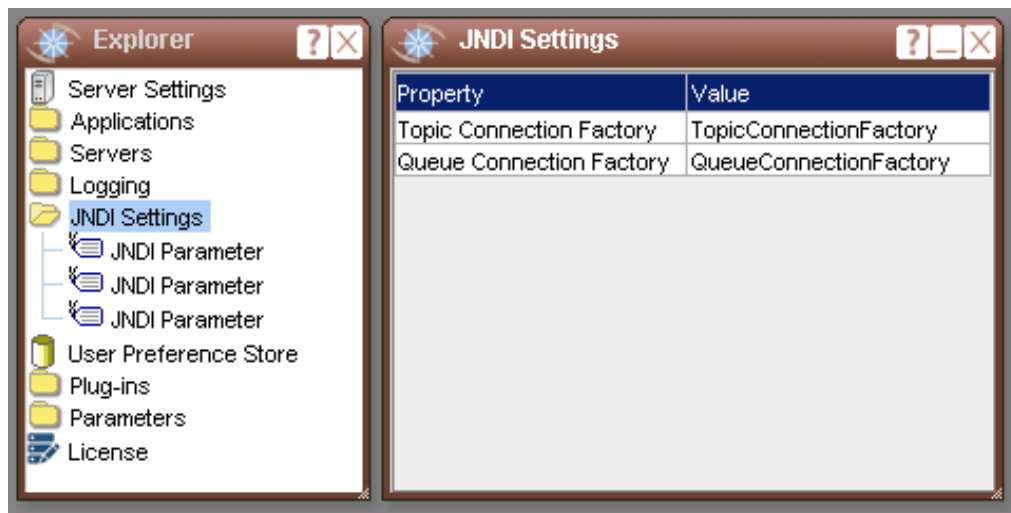
When AltioLive is started (Start Tomcat), you should be able to open the debug tool and find the following in the Server Log Files (log_XXXX.txt file) – if you can't find it, up the KBytes shown. The key information (indicating that JNDI and JMS are available) is highlighted in bold:

```
11:38:06>AltioSyncEngine [INFO] Generating checksum for file C:\installs\altiolive_studio54\tomcat\webapps\altio54\WEB-INF\classes\defaults\skins\al_helpskin.xml
11:38:06>AltioSyncEngine [INFO] Checksums correct
```

```
11:38:06>AltioSyncEngine [INFO] -- Start Jndi --
11:38:07>AltioSyncEngine [INFO] JNDI: entry (java.naming.factory.initial) value (org.jnp.interfaces.NamingContextFactory)
11:38:07>AltioSyncEngine [INFO] JNDI: entry (java.naming.provider.url) value (localhost:1099)
11:38:07>AltioSyncEngine [INFO] JNDI: entry (java.naming.factory.url.pkgs) value (org.jboss.naming)
11:38:07>AltioSyncEngine [INFO] JNDI: TopicConnectionFactory located under: TopicConnectionFactory
11:38:07>AltioSyncEngine [INFO] JNDI: QueueConnectionFactory located under: QueueConnectionFactory
11:38:07>AltioSyncEngine [INFO] JNDI: created initial context, about to connect...
11:38:07>AltioSyncEngine [INFO] JNDI: connected to naming service!
11:38:07>AltioSyncEngine [INFO] JNDI: available
11:38:07>AltioSyncEngine [INFO] -- End Jndi --
11:38:07>AltioSyncEngine [INFO] JMS available
```

JNDI Settings

JNDI settings are configured in the **Server Administration Tool**. The default settings are configured for a local installation of JBoss 2.



JNDI Access via the Global Context

To access JNDI via the Global Context (a different context from AltioLive), set properties as follows

Object	Property	Value
JNDI Settings	Topic Connection Factory	TopicConnectionFactory
	Queue Connection Factory	QueueConnectionFactory
JNDI Parameter (1)	Name	java.naming.factory.initial
	Value	org.jnp.interfaces.NamingContextFactory
JNDI Parameter (2)	Name	java.naming.provider.url
	Value	localhost:1099
JNDI Parameter (3)	Name	java.naming.factory.url.pkgs
	Value	org.jboss.naming

Topic Connection Factory

Used with JMS, for publish-and-subscribe messaging. Before using any topics in JMS, you must first create a connection to a topic using a topic connection factory. This object is usually located in the JNDI namespace under the name "TopicConnectionFactory". This property allows you to change the name that the topic connection factory is identified by in the naming service.

Queue Connection Factory

Used with JMS, for point-to-point messaging. In JMS before using any queues, you must first create a connection to a queue using a queue connection factory. This object is normally located in the JNDI namespace under the name "QueueConnectionFactory", but is not always so.

The parameters (name/value pairs) map directly to the hashtable that is used to configure the InitialContext access to the naming service. Additional JNDI parameters can be added and set as required to interface with a particular J2EE implementation.

Using JMS with Altio

Overview

JMS can be used for Service Functions and for Datapools. JMS Service Functions cannot be used to get data on demand as they return no data. Similarly, they cannot be used for initial data.

Service Functions would want to be able to send to either queues or topics. For Datapools we only require subscribing to topics as being a queue subscriber does not seem appropriate (queues are often used for example for load balancing).

There are four formats available for messages (both topics and queues):

- XML
- Name-value pairs
- Text
- Binary messages

However, AltioLive currently only supports Text.

AltioLive also does not support setting headers or filtering.

Configuring a Service Function

In the Application Manager, there is a window for configuring JMS services:

JNDI Name - the name in the JNDI namespace of the topic or queue to send the message to.

Message Body - Enter the text of the message, which may contain one or more references to the data from the client (e.g. `${client.PROD}`) as well as general session and property references.

JMS Service JMS SERVICE1

Java Messaging Service

Service name: JMS SERVICE1

JNDI name: topic/newSaleService

Message body:

```
<DATA QTY=${client.QTY} PROD_ID=${client.PROD_ID} PROD=${client.PROD}
CTRY=${client.CTRY} PORT=${client.PORT} CURRENCY=${client.CURRENCY}
AL_NAME='SALE_ITEM' APPID=${application.id} SELLER_ID=${connection.user}
PRICE=${client.PRICE} COMMENT=${client.COMMENT} EXP_DATE=${client.EXP_DATE}
ACTION='NEW' GUID='SALE_ID' TARGET=/FOR_SALE' SH_TRMS=${client.SH_TRMS}/>
```

Acknowledgement

	Source	Text
On success	DEFINE <input checked="" type="checkbox"/>	Your new sale item notification has been sent
On failure	DEFINE <input checked="" type="checkbox"/>	Your new sale item notification has been sent

Offline Configuration... OK Cancel

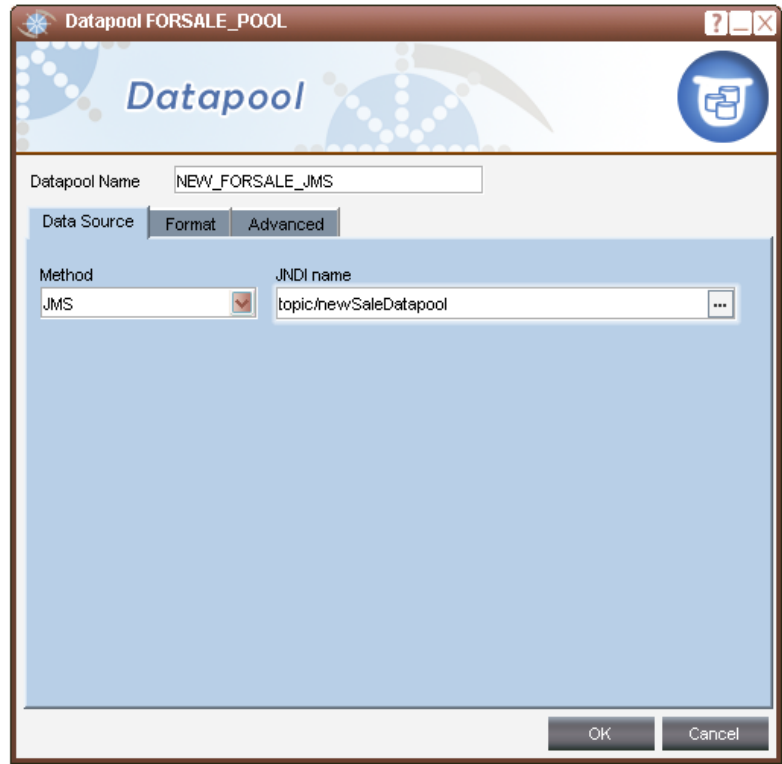
Configuring a Datapool

Datapools can register as listeners to JMS topics (set Method in the dropdown list to JMS), and will receive all messages sent to that topic.

- The messages must contain well formed XML
- The timestamp must be on the root node of the data.

JNDI Name - the name under which the topic is bound (must be a topic as opposed to a queue)

Using JMS for datapools is efficient with network resources as data is pushed to the datapool instead of being pulled in by polling.

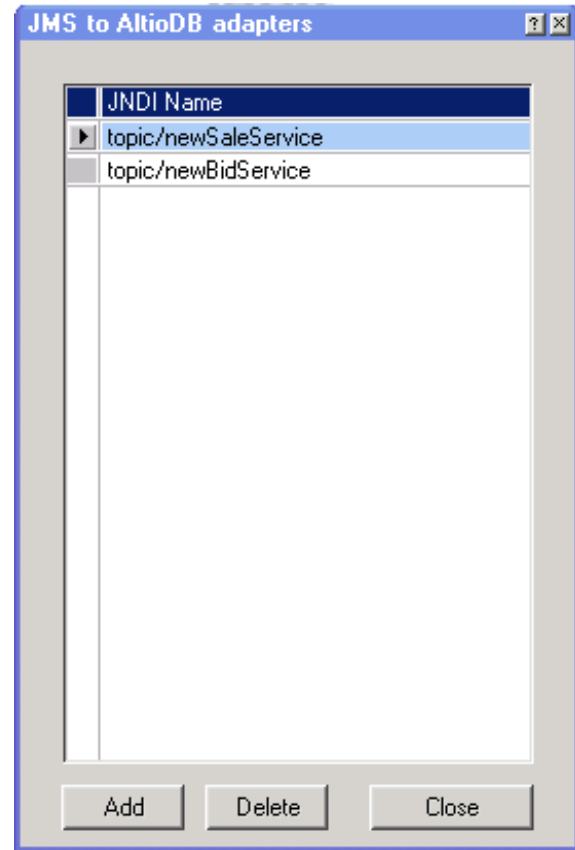


Configuring AltioDB

AltioDB provides basic JMS support – primarily aimed at demonstrating how AltioLive can work with JMS. Via the AltioDB Manager tool, you can configure support for JMS.

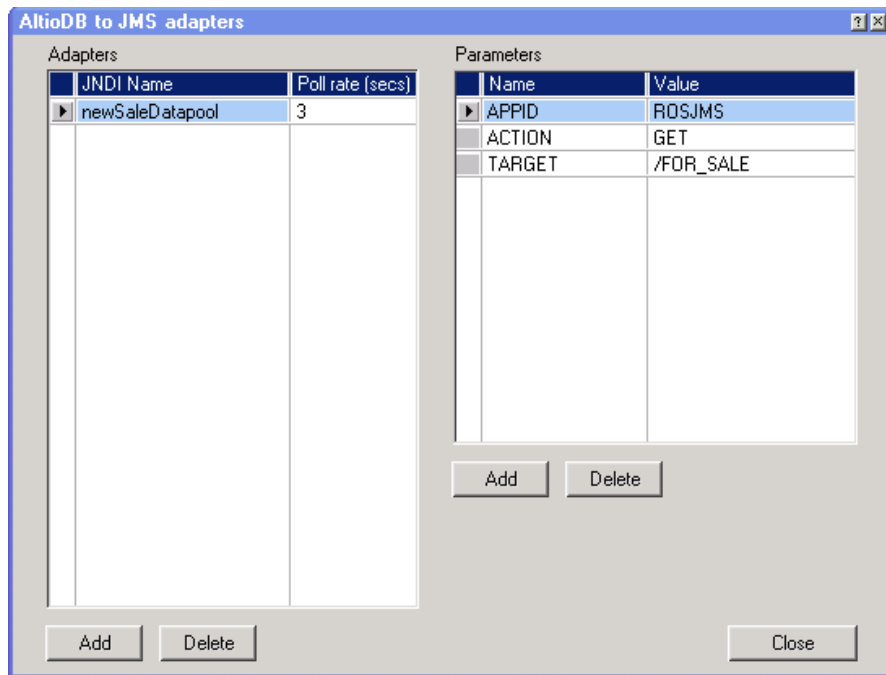
JMS to AltioDB Adapters allow AltioDB to receive updates via JMS. JMS adapters register themselves as a subscriber to the specified topic and pass any messages sent to that topic into AltioDB. The message it expects is as text – comprised of a well-formed XML element. The attributes from that element are used as the name-value pairs that are passed to AltioDB.

This window names the adapters/topics that AltioDB will listen to.



AltioDB to JMS Adapters allow AltioDB to send updates via JMS. The adapter will regularly poll AltioDB, and if there is any new data, return it as a JMS message to the specified topic.

This window names the topics that AltioDB will send data to, and specifies the XPath query (Target) to use for getting data updates. AltioDB automatically takes care of timestamp information.



Configuring ROS (JMS) Techlet

The application works the same as standard ROS, except where service functions and datapools have been replaced with JMS alternatives. The main thing to keep in mind is that the service functions (new sale and new bid) now just send a message. All we know is that the message was received by JMS. Only via the datapool is there any indication that the service function was executed successfully.

In the following, we demonstrate how to reference a JMS Queue instead of a JMS Topic.

Configure JBoss

You can use the JBoss 2 console to configure a JMS Queue as indicated below (you will need to unregister the newBidService Topic first):

Agent Administration [JMX RI1.0]

[Back to Agent View](#)

Specify the object name and java class of the MBean to add, delete or view the constructors of:
(Optionally provide a class loader name for loading the specified class.)

Domain: <input type="text" value="JBossMQ"/>
Keys: <input type="text" value="service=Queue,name=newBidService"/>
Java Class: <input type="text" value="org.jboss.mq.server.QueueManager"/>
Class Loader: <input type="text"/>

Action:

Once the queue is defined, if you click on the queue in the “Agent View”, to bring up the MBean View:

○ **JBossMQ**

- [service=InvocationLayer,type=JVM](#)
- [service=InvocationLayer,type=OIL](#)
- [service=InvocationLayer,type=RMI](#)
- [service=InvocationLayer,type=UTIL](#)
- [service=PersistenceManager](#)
- [service=Queue,name=A](#)
- [service=Queue,name=B](#)
- [service=Queue,name=C](#)
- [service=Queue,name=D](#)
- [service=Queue,name=E](#)
- [service=Queue,name=F](#)
- [service=Queue,name=controlQueue](#)
- [service=Queue,name=ex](#)
- [service=Queue,name=myQueue](#)
- [service=Queue,name=newBidService](#)
- [service=Queue,name=testQueue](#)
- [service=Server](#)
- [service=StateManager](#)
- [service=Topic,name=bob](#)
- [service=Topic,name=example](#)
- [service=Topic,name=newSaleDatapool](#)
- [service=Topic,name=newSaleService](#)
- [service=Topic,name=testTopic](#)

In the MBean View scroll down and click on the init button to start the queue (this seems to be managed automatically when JBoss is started).

List of MBean operations:

Description of destroy

void `destroy`

Description of stop

void `stop`

Description of init

void `init`

Or, you can replace the newBidService entry in the jboss.jcml file as follows, and restart JBoss:

```
<mbean code="org.jboss.mq.server.QueueManager" name="JBossMQ:service=Queue,name=newBidService"/>
```

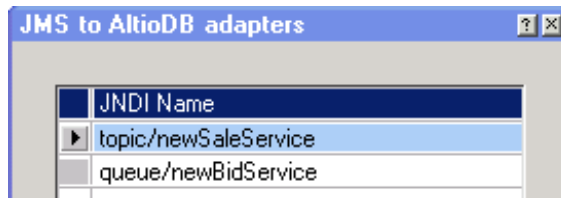
Configure Altio

Change the JNDI name in the ROSJMS Integration XML to be queue/newBidService:



Configure AltioDB

Change the JNDI name in the JMS to AltioDB adapters to be queue/newBidService:



Running ROSJMS

Run the application as before ...

ENC Configuration

AltioLive can also be deployed in the J2EE environment running JBoss. In this scenario, all the all the JNDI Settings should be blanked (you cannot delete them) and all the JNDI Parameters should be deleted. This means that JNDI is accessed without any parameters, which forces it to pick up the ENC.

The information available via the ENC is controlled by the information in the web.xml file (located under WEB-INF in the web application). If the J2EE server you are using has a graphical interface, then this information can usually be set from there. The names used to access retrieve objects from the ENC are normally of the form "java:comp/env/jms/myTopic".

Deploying Altio

1. Copy the altio54.war file to the tomcat embedded with JBoss
2. Start JBoss with Tomcat – should automatically deploy the war file
3. Edit the altioserver.xml file to have the same port as the tomcat supplied with JBoss (8080)
4. Put rosjms.aar file in deploy directory
5. Restart JBoss
6. Open SE Admin tool and clear JNDI settings
7. Restart JBoss (to reset JNDI settings)
8. Run ROSJMS application

Document Information

Integra SP – Altio

Telephone: +44 (0) 20 8528 1045 Internet: www.altio.com

Copyright © 2009 Integra SP

Copyright in this document is vested in Integra SP. The contents of the document (wholly or in part) must not be reproduced, distributed, used or disclosed without the prior written permission of Integra SP.

Integra recognizes the trademarks or registered trademarks of any third party product or company name referenced in this document at the time of its publication.