JBoss Enterprise SOA Platform 5.0

ESB Administration Guide

Your guide to administering the JBoss SOA Platform ESB

JBoss Enterprise SOA Platform 5.0 ESB Administration Guide
Your guide to administering the JBoss SOA Platform ESB Edition 3.0

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

The Administration Guide contains important information on how to configure and manage the ESB of the JBoss Enterprise SOA Platform 5.0.

Preface

1. Document Conventions

This manual uses several conventions to highlight certain words and phrases and draw attention to specific pieces of information.

In PDF and paper editions, this manual uses typefaces drawn from the *Liberation Fonts*[1] set. The Liberation Fonts set is also used in HTML editions if the set is installed on your system. If not, alternative but equivalent typefaces are displayed. Note: Red Hat Enterprise Linux 5 and later includes the Liberation Fonts set by default.

1.1. Typographic Conventions

Four typographic conventions are used to call attention to specific words and phrases. These conventions, and the circumstances they apply to, are as follows.

`Mono-spaced Bold`

Used to highlight system input, including shell commands, file names and paths. Also used to highlight key caps and key-combinations. For example:

> To see the contents of the file **`my_next_bestselling_novel`** in your current working directory, enter the **`cat my_next_bestselling_novel`** command at the shell prompt and press **`Enter`** to execute the command.

The above includes a file name, a shell command and a key cap, all presented in Mono-spaced Bold and all distinguishable thanks to context.

Key-combinations can be distinguished from key caps by the hyphen connecting each part of a key-combination. For example:

> Press **`Enter`** to execute the command.
>
> Press **`Ctrl`**+**`Alt`**+**`F1`** to switch to the first virtual terminal. Press **`Ctrl`**+**`Alt`**+**`F7`** to return to your X-Windows session.

The first sentence highlights the particular key cap to press. The second highlights two sets of three key caps, each set pressed simultaneously.

If source code is discussed, class names, methods, functions, variable names and returned values mentioned within a paragraph will be presented as above, in **`Mono-spaced Bold`**. For example:

> File-related classes include **`filesystem`** for file systems, **`file`** for files, and **`dir`** for directories. Each class has its own associated set of permissions.

Proportional Bold

This denotes words or phrases encountered on a system, including application names; dialogue box text; labelled buttons; check-box and radio button labels; menu titles and sub-menu titles. For example:

[1] https://fedorahosted.org/liberation-fonts/

Choose **System > Preferences > Mouse** from the main menu bar to launch **Mouse Preferences**. In the **Buttons** tab, click the **Left-handed mouse** check box and click **Close** to switch the primary mouse button from the left to the right (making the mouse suitable for use in the left hand).

To insert a special character into a **gedit** file, choose **Applications > Accessories > Character Map** from the main menu bar. Next, choose **Search > Find…** from the **Character Map** menu bar, type the name of the character in the **Search** field and click **Next**. The character you sought will be highlighted in the **Character Table**. Double-click this highlighted character to place it in the **Text to copy** field and then click the **Copy** button. Now switch back to your document and choose **Edit > Paste** from the **gedit** menu bar.

The above text includes application names; system-wide menu names and items; application-specific menu names; and buttons and text found within a GUI interface, all presented in Proportional Bold and all distinguishable by context.

Note the **>** shorthand used to indicate traversal through a menu and its sub-menus. This is to avoid the difficult-to-follow 'Select **Mouse** from the **Preferences** sub-menu in the **System** menu of the main menu bar' approach.

Mono-spaced Bold Italic or ***Proportional Bold Italic***

Whether Mono-spaced Bold or Proportional Bold, the addition of Italics indicates replaceable or variable text. Italics denotes text you do not input literally or displayed text that changes depending on circumstance. For example:

To connect to a remote machine using ssh, type **ssh *username*@*domain.name*** at a shell prompt. If the remote machine is **example.com** and your username on that machine is john, type **ssh john@example.com**.

The **mount -o remount *file-system*** command remounts the named file system. For example, to remount the **/home** file system, the command is **mount -o remount /home**.

To see the version of a currently installed package, use the **rpm -q *package*** command. It will return a result as follows: ***package-version-release***.

Note the words in bold italics above — username, domain.name, file-system, package, version and release. Each word is a placeholder, either for text you enter when issuing a command or for text displayed by the system.

Aside from standard usage for presenting the title of a work, italics denotes the first use of a new and important term. For example:

When the Apache HTTP Server accepts requests, it dispatches child processes or threads to handle them. This group of child processes or threads is known as a *server-pool*. Under Apache HTTP Server 2.0, the responsibility for creating and maintaining these server-pools has been abstracted to a group of modules called *Multi-Processing Modules* (*MPMs*). Unlike other modules, only one module from the MPM group can be loaded by the Apache HTTP Server.

1.2. Pull-quote Conventions

Two, commonly multi-line, data types are set off visually from the surrounding text.

Output sent to a terminal is set in `Mono-spaced Roman` and presented thus:

```
books        Desktop   documentation  drafts  mss    photos   stuff  svn
books_tests  Desktop1  downloads      images  notes  scripts  svgs
```

Source-code listings are also set in `Mono-spaced Roman` but are presented and highlighted as follows:

```
package org.jboss.book.jca.ex1;

import javax.naming.InitialContext;

public class ExClient
{
   public static void main(String args[])
       throws Exception
   {
      InitialContext iniCtx = new InitialContext();
      Object         ref    = iniCtx.lookup("EchoBean");
      EchoHome       home   = (EchoHome) ref;
      Echo           echo   = home.create();

      System.out.println("Created Echo");

      System.out.println("Echo.echo('Hello') = " + echo.echo("Hello"));
   }

}
```

1.3. Notes and Warnings

Finally, we use three visual styles to draw attention to information that might otherwise be overlooked.

Note

A note is a tip or shortcut or alternative approach to the task at hand. Ignoring a note should have no negative consequences, but you might miss out on a trick that makes your life easier.

Important

Important boxes detail things that are easily missed: configuration changes that only apply to the current session, or services that need restarting before an update will apply. Ignoring Important boxes won't cause data loss but may cause irritation and frustration.

Warning
A Warning should not be ignored. Ignoring warnings will most likely cause data loss.

2. We Need Feedback!

If you find a typographical error in this manual, or if you have thought of a way to make this manual better, we would love to hear from you! Please submit a report in Bugzilla: *http://bugzilla.redhat.com/bugzilla/* against the product **JBoss Enterprise SOA Platform.**

When submitting a bug report, be sure to mention the manual's identifier: *SOA_ESB_Administration_Guide*

If you have a suggestion for improving the documentation, try to be as specific as possible when describing it. If you have found an error, please include the section number and some of the surrounding text so we can find it easily.

Configuration

1.1. Stand-Alone Server

If you wish to run the JBoss SOA-P Enterprise Service Bus server on the same machine as the *JBoss Application Server* (JBossAS), then you should study the information on this website: *http://wiki.jboss.org/wiki/ConfiguringMultipleJBossInstancesOnOneMachine*.

1.2. Clustered ESB Service

If one is using a clustered ESB environment, the nodes must be configured in one of the following two ways:

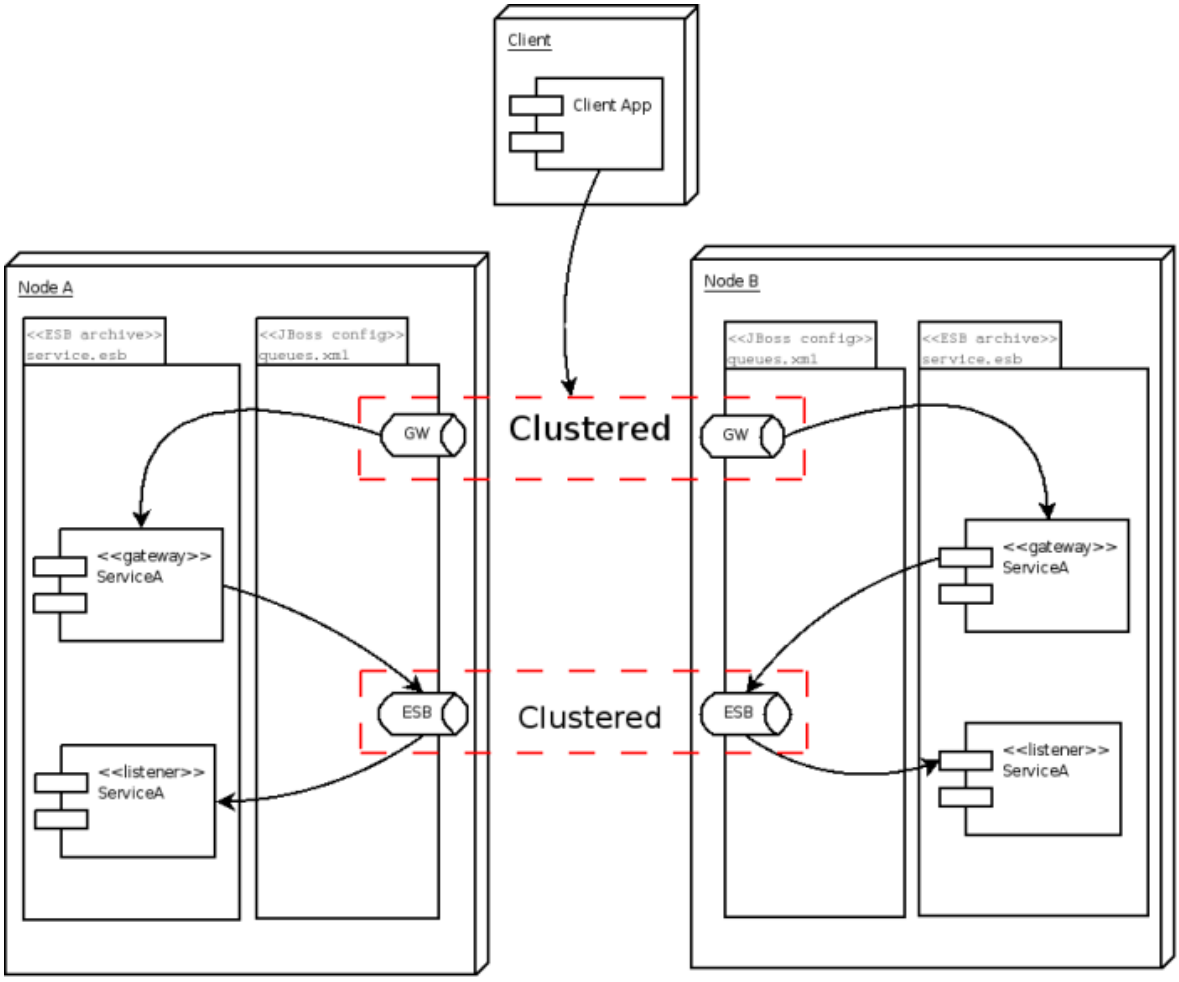

Figure 1.1. Clustering Scenario One

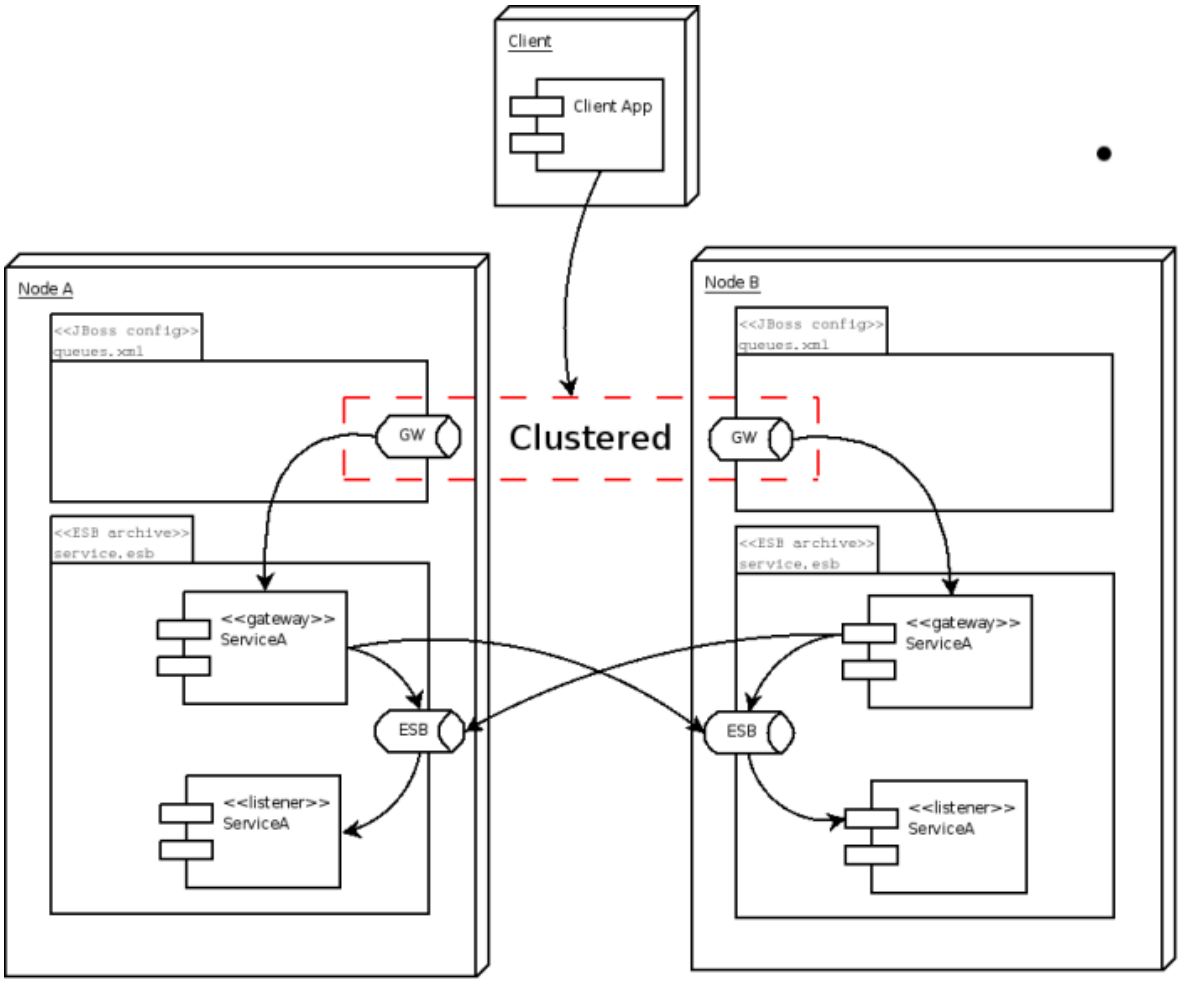

Figure 1.2. Clustering Scenario Two

Two other points that must be taken into account are:

- All of the clustered queues must be deployed on each and every one of the cluster nodes at all times.
- The client must list all of the nodes in the **`jndi.properties`** file as per this example code:

```
java.naming.provider.url=jnp://jawa01:1099
java.naming.provider.url=jnp://jawa02:1099
```

1.3. *Updated.* JBossESB Java Message Service Providers

Red Hat currently support the use of a number of *Java Message Service* applications in conjunction with the JBoss Enterprise SOA-P. These applications are *JBoss Messaging*, *Tibco* 5.0, and *IBM MQ* 6.0.2.

Red Hat recommends that you use JBoss Messaging; it is included with the default configuration.

Any *JSR-914* (*http://jcp.org/en/jsr/detail?id=914*)-compliant *Java Message Service* implementation (such as *Apache ActiveMQ* or *OracleAQ*) should also work . However, other JMS providers have not been fully tested and are not supported at this time. If you wish to try another vendor's product, you should consult their documentation.

Important

This section is not intended to be a replacement for the configuration documentation that comes with the supported Java Message Service implementations. For advanced capabilities, such as clustering and management, you should consult that documentation as well.

How Can I Configure Them?

You can configure **JMSListeners** and **JMSGateways** to listen to either "Queues" or "Topics" by setting the following parameters in the **Service Configuration** file:

- jndi-URL
- jndi-context-factory
- jndi-pkg-prefix
- connection-factory
- destination-type
- destination-name

You will need to ensure that your chosen Java Message Service provider's client **JAR** files are included in your classpath.

Each **JMSListener** and **JMSGateway** can be configured to use its own Java Message Service provider, allowing you to use more than one provider in your deployment if you so desire.

The Service-Oriented Architecture Platform utilises a connection pool to improve performance when a Java Message Service is in use. By default, the size of this pool is set to twenty, but this figure can be over-ridden if one reconfigures the org.jboss.soa.esb.jms.connectionPool property in the `transports` section of the ESB configuration file. The service will keep re-trying for up to thirty seconds if an initial session cannot be obtained. This time-out period can be configured by changing the value associated with the org.jboss.soa.esb.jms.sessionSleep property.

Maximum Sessions Per Connection

The **JBoss** Enterprise Service Bus' **JmsConnectionPool** collates **Java** Message Service sessions. It is used by all JMS-based components, including JMS Listeners, JMS Couriers and the JMS Router.

Some providers limit the number of Java Message Service Sessions per connection. This means that the JMS components in the JBoss Enterprise Service Bus are required to support a mechanism that controls the maximum number of sessions created from each connection that is managed by a single `JmsConnectionPool` instance.

The components achieve this by using one or both of of the following properties specified in their own **JNDI** configuration (whether it be the JMS Provider/Bus, **JMSRouter** or so on):

- **org.jboss.esb.jms.max.sessions.per.connection**: This is the maximum total number of sessions allowed per connection (for XA and non-XA session instances.) The default value of twenty is equal to that of the maximum number of Java Message Service sessions allowed for a whole `JmsConnectionPool`. (This is configured in the **jbossesb-properties.xml** file).

- **`org.jboss.esb.jms.max.xa.sessions.per.connection`**: This is the maximum number of **XA** sessions allowed per connection (**XA** only.) This value equals that of **`org.jboss.esb.jms.max.sessions.per.connection`** by default.

If neither of the above parameters are configured, the **`JmsConnectionPool`** will create a single Java Message Service connection. All JMS sessions will, in turn, be created from that connection instance.

One should configure these settings as generic properties. To do so, use the Java Message Service provider configuration:

```
<jms-provider ...>
    <property
        name="org.jboss.esb.jms.max.sessions.per.connection" value="5" />
    <property
        name="org.jboss.esb.jms.max.xa.sessions.per.connection" value="1" /
>
    <!-- And add providers.... -->
</jms-provider>
```

Important

In the sections that follow, these assumptions are made:

- your JMS provider runs on 'localhost'
- the connection factory is called '`ConnectionFactory`'
- the destination-type is a 'queue'
- the destination-name is 'queue/A'

1.3.1. *Updated JBoss Messaging*

JBoss Messaging is the default JMS provider for the JBoss SOA Platform.

One should set the parameters for JBoss Messaging to:

```
jndi-URL="localhost”
jndi-context-factory="org.jnp.interfaces.NamingContextFactory"
connection-factory="ConnectionFactory"
destination-type="queue"
destination-name="queue/A"
```

The JAR file called **`jboss-messaging-client.jar`** must be included in your class path. Note that this JAR file is included in the **`jbossall-client.jar`** archive, which can be found in the **`lib/ext`** directory.

Instructions for installing JBoss Messaging can be found on the project website:

http://labs.jboss.com/jbossmessaging/docs/userguide-1.4.0.GA/html/installation.html

1.3.1.1. *Updated* JBoss Messaging Clustering Configuration

By configuring JBoss Messaging in a clustered set up, you will gain load balancing and fail-over facilities for JMS. Note that this capability is continually evolving between different versions of JBoss Messaging, so you should consult the relevant JBoss Messaging documentation to learn about the latest features of this program.

1.3.2. *Apache ActiveMQ*

Warning

Apache ActiveMQ has not been fully tested & is not a supported JMS implementation.

You should set the parameters for Apache ActiveMQ to:

```
jndi-URL="tcp://localhost:61616”
jndi-context-
factory="org.apache.activemq.jndi.ActiveMQInitialContextFactory"
connection-factory="ConnectionFactory"
destination-type="queue"
destination-name="queue/A"
```

In your classpath you should have:

- **`activemq-core-4.x`**
- **`backport-util-concurrent-2.1.jar`**

Both of these JARs can be found in the **`lib/ext/jms/activemq`** file.

Apache ActiveMQ has been tested with version 4.1.0-incubator.

1.3.3. *Updated.* IBM Websphere MQ Series

You should set the main JNDI parameters for the IBM Websphere MQ Series to:

```
jndi-URL="localhost:1414/SYSTEM.DEF.SVRCONN”
jndi-context-factory="com.ibm.mq.jms.context.WMQInitialContextFactory"
connection-factory="WMQQueueManager"
destination-type="queue"
destination-name="QUEUEA"
```

Note

You will also need to configure the **`max-xa-sessions-per-connection`** property to a value of “1” if you use **`XA`** Connections. For more information on how to configure this property, see the "Max. Sessions Per Connection" section of this chapter.

Extra WMQ JAR files

Note that the connection-factory setting should refer to the name of the WMQ Queue Manager on the WMQ Server on which the Java Message Service destination is configured.

In order to be able to connect to a WMQ Provider from the JBoss Enterprise Service Bus, you will need to add some additional JAR files to your ***${SOA_ROOT}*/server/*${CONFIG}*/lib/** directory:

One should have the following on one's classpath':

- **com.ibm.mq.pcf.jar**
- **mqcontext.jar**
- **com.ibm.mq.jar** (client JAR)
- **com.ibm.mqjms.jar** (client JAR)

Note that the client JARs are not open source and are not provided by Red Hat. One will have to obtain them from one's Websphere Application Server and MQ installations.

1.3.4. XA Connections

If one needs to manage one's XA Connection Resources on the JBoss Enterprise Service Bus using WMQ, one will need to install the WMQ Extended Client JAR file in the ***${SOA_ROOT}*/server/*${CONFIG}*/lib/** directory. This JAR file is typically named **com.ibm.mqetclient.jar** and must be acquired from your IBM partner or agent.

Once this library is installed, WMQ restricts the number of Java Message Service Sessions per JMS Connection to one. This restriction applies to both XA and non-XA WMQ JMS Connections. In this event, you will need to configure the **org.jboss.esb.jms.max.sessions.per.connection** property of the **JmsConnectionPool** associated with the your WMQ JMS components (JMS Listeners, JMSRouter and so forth.) Simply set the value of this property to 1. See the "Max Sessions Per Connection" section of this document for further information.

The obvious implication of setting this property is that your **JmsConnectionPool** will consume more Java Message Service connections on your WMQ Provider. Note that this setting does not need to be configured for any WMQ-based JCA provider configurations.

1.3.5. JCA Adapter

The default installation of IBM Websphere MQ Version 6.0.2 does not include the WMQ *JCA Adapter*.

Procedure 1.1. Setting up the WMQJCA Adapter

1. **Update IBM Websphere MQ Version 6.0.2**
   You will need to update your Websphere MQ installation to at least version 6.0.2.1 in order to obtain the JCA Adapter.

2. **Deploy the Adapter on the SOA-P Server**
   The WMQ JCA Adapter, (**wmq.jmsra.rar**), is found in the **Java/lib/jca/** directory of your WebSphere MQ installation. To deploy the adapter, copy this file to the **deploy** directory of your SOA-P server, **${SOA_ROOT}/server/${CONFIG}/deploy/**.

3. **How to Create a JCA Connection Factory Configuration**

   To configure the SOA-P server to use the WMQJCA Adapter, one must create a JCA*Connection Factory* configuration file. The filename does not strictly matter but it should be descriptive, such as **`wsmq-jms-ds.xml`**. This configuration has to be copied into the deploy directory of the SOA-P server, which is as follows: **`${SOA_ROOT}/server/{CONFIG}/deploy/`**.

```
<?xml version="1.0" encoding="UTF-8"?>
<connection-factories>

<mbean code="org.jboss.jms.jndi.JMSProviderLoader"
    name=":service=JMSProviderLoader,name=WSMQJmsProvider">
    <attribute name="ProviderName">WSMQProvider</attribute>
    <attribute name="ProviderAdapterClass">
        org.jboss.jms.jndi.JNDIProviderAdapter
    </attribute>
    <attribute name="QueueFactoryRef">ConnectionFactory</attribute>
    <attribute name="TopicFactoryRef">ConnectionFactory</attribute>
    <attribute name="FactoryRef">ConnectionFactory</attribute>
    <attribute name="Properties">
java.naming.factory.initial=com.ibm.mq.jms.context.WMQInitialContextFactory
java.naming.provider.url=mqserver.domain.com:1414/SYSTEM.DEF.SVRCONN
    </attribute>
</mbean>

<tx-connection-factory>
    <jndi-name>WSMQJmsXA</jndi-name>
    <xa-transaction/>
    <rar-name>jms-ra.rar</rar-name>
    <connection-definition>
        org.jboss.resource.adapter.jms.JmsConnectionFactory
    </connection-definition>
    <config-property name="SessionDefaultType" type="java.lang.String">
        javax.jms.Queue
    </config-property>
    <config-property name="JmsProviderAdapterJNDI"
 type="java.lang.String">
        java:/WSMQProvider
    </config-property>
    <max-pool-size>20</max-pool-size>
    <security-domain-and-application>
        JmsXARealm
    </security-domain-and-application>
</tx-connection-factory>

</connection-factories>
```

Example 1.1. JCA Connection Factory Configuration

The JCA Adapter is now available and can be accessed via the configured connection factory.

In your Enterprise Server Bus configuration, you can now use this connection factory through the JCA Provider configuration to provide transactional context. The following example does so in this file: **${SOA_ROOT}/server/${CONFIG}/deploy/jbpm.esb/META-INF/jboss-esb.xml**.

```
<?xml version="1.0" encoding="UTF-8"?>
<jbossesb xmlns="http://anonsvn.labs.jboss.com/labs/jbossesb/trunk/product/
etc/schemas/xml/jbossesb-1.0.1.xsd" parameterReloadSecs="5">

<providers>
  <jms-jca-provider connection-factory="ConnectionFactory"
    jndi-URL="mqserver.domain.com:1414/SYSTEM.DEF.SVRCONN"
    jndi-context-factory="com.ibm.mq.jms.context.WMQInitialContextFactory"
    name="CallbackQueue-JMS-Provider"
    providerAdapterJNDI="java:/WSMQProvider">
    <jms-bus busid="jBPMCallbackBus">
      <jms-message-filter dest-name="queue/CallbackQueue"
        dest-type="QUEUE"/>
    </jms-bus>
  </jms-jca-provider>
</providers>

<services>
  <service category="JBossESB-Internal"
    description="Service which makes Callbacks into jBPM"
    name="JBpmCallbackService">
    <listeners>
      <jms-listener busidref="jBPMCallbackBus" maxThreads="1"
        name="JMS-DCQListener"/>
    </listeners>
    <actions mep="OneWay">
      <action
        class="org.jboss.soa.esb.services.jbpm.actions.JBpmCallback"
        name="action"/>
    </actions>
  </service>
</services>

</jbossesb>
```

1.3.6. Authentication

Also note that one may see the following exception error message when running MQ 6.0.2. It can be readily fixed by adding the name of the user that runs the JBoss Enterprise SOA Platform to the mqm group.

```
Message: Unable to get a MQ series Queue Manager or Queue Connection.
 Reason: failed to
create connection -javax.jms.JMSSecurityException: MQJMS2013: invalid
 security
authentication supplied for MQQueueManager
```

1.3.7. *Oracle Advanced Queuing (AQ)*

Warning

Oracle AQ has not been fully tested and it is not a supported JMS implementation.

One should set the parameters for Oracle AQ to the following:

```
connection-factory=”QueueConnectionFactory”
```

and then use the following properties to configure it correctly:

```
<property name="java.naming.factory.initial"
    value="org.jboss.soa.esb.oracle.aq.AQInitialContextFactory"/>
<property name="java.naming.oracle.aq.user" value="<user>"/>
<property name="java.naming.oracle.aq.password" value="<pw>"/>
<property name="java.naming.oracle.aq.server" value="<server>"/>
<property name="java.naming.oracle.aq.instance" value="<instance>"/>
<property name="java.naming.oracle.aq.schema" value="<schema>"/>
<property name="java.naming.oracle.aq.port" value="1521"/>
<property name="java.naming.oracle.aq.driver" value="thin"/>
```

Alternatively, one could specify a database connection uniform resource locator. The following example demonstrates how to allow an user to connect to an Oracle "Real Application Cluster" (RAC):

```
<property name="java.naming.factory.initial"
 value="org.jboss.soa.esb.oracle.aq.AQInitialContextFactory"/>
<property name="java.naming.oracle.aq.user"     value="<user>"/>
<property name="java.naming.oracle.aq.password" value="<pw>"/>
<property name="java.naming.oracle.aq.url"
 value="jdbc:oracle:thin:@(description=(address_list=(load_balance=on)
(failover=on)(address=(protocol=tcp)(host=host1)
(port=1621))(address=(protocol=tcp)(host=host2)(port=1621)))
(connect_data=(service_name=SID)(failover_mode=(type=select)
(method=basic)))) "/>
```

Note the reference to the `InitialContext factory`. One only needs this if one wants to avoid having OracleAQ register its queues with a lightweight directory access protocol server. The **`AqinitialContextFactory`** refers to code in a plug-in JAR that one can find in the **`plugins/org.jboss.soa.esb.oracle.aq`** directory. The JAR file is called **`org.jboss.soa.esb.oracle.aq-4.2.jar`** and one will have to deploy it to the **`jbossesb.sar/lib`** directory.

When one creates a queue in Oracle AQ, it must be ensured that a payload type of `SYS AQ $_JMS_MESSAGE` is selected.

An example **`jboss-esb.xml`** configuration file that one may find useful can be found in the **`samples/quickstarts/helloworld_action/oracle-aq`** directory.

1.3.8. *Tibco Enterprise Message Service* (EMS)

One should set the parameters for the Tibco Enterprise Message Service to the following:

```
jndi-URL="tcp://localhost:7222”
jndi-context-factory=”com.tibco.tibjms.naming.TibjmsInitialContextFactory"
connection-factory="QueueConnectionFactory"
destination-type="queue"
destination-name="<queue-name>"
```

Check that the client JARs that ship with Tibco EMS can be found in the classpath. These files are located in the **tibco/ems/clients/java** directory.

- **jaxp.jar**
- **jndi.jar**
- **tibcrypt.jar**
- **tibjmsapps.jar**
- **tibrvjms.jar**
- **jms.jar**
- **jta-spec1_0_1.jar**
- **tibjmsadmin.jar**
- **tibjms.jar**

Red Hat has tested version 5.0 of the Tibco EMS with the JBoss Service-Oriented Architecture Platform.

1.3.9. Extension Properties

By default, the JNDI configuration used to retrieve the JMS resources is configured to inherit all of those properties with names prefixed with "java.naming". Note that some Java Message Service providers may specify properties that use a different naming prefix.

In order to support these different schemes, Red Hat provides a mechanism to specify individual property prefixes for each provider. This functionality allows the software to inherit properties which use these additional prefixes.

One configures the prefixes by defining the "jndi-prefixes" property on its associated jms-provider element with a comma-separated list of the additional prefixes. The extension properties are also configured in this same location.

```
<jms-provider name=”JMS” connection-factory=”ConnectionFactory”>
    <property name=”jndi-prefixes” value=”test.prefix.”/>
    <property name=”test.prefix.extension1” value=”extension1”/>
    <property name=”test.prefix.extension2” value=”extension2”/>
</jms-provider>
```

1.3.10. *Updated* *File Transfer Protocol* (FTP) Configuration

Most configuration options are set via the File Transfer Protocol *Endpoint Reference* (EPR). They are described in the *Programmers' Guide*. However, configuration data is set at the global level in the **`jbossesb-properties`** file.

`org.jboss.soa.esb.ftp.renameretry`

The JBoss Enterprise Server Bus renames all files to have the same name prior to transmitting them via File Transfer Protocol (This prevents the individual files from being processed until the program then renames them at the other end.) Unfortunately, some File Transfer Protocol servers retain a lock on the file after it has been written, which prevent the renaming process from taking place. If this problem occurs, **JBossESB** will make the defined number of attempts (the default being ten) to rename the files, "sleeping" in between, until it finally generates an error message if the file cannot be renamed.

1.3.11. *Updated* Database Configuration

The Service-Oriented Architecture Platform uses both a database and the *Message Store* in order to keep registry services "persistent." The JBoss SOA-P server includes the *Hypersonic* database (HSQLDB). The data-source definition can be found in this location: ***`${SOA_ROOT}`*`/server/`*`${CONFIG}`*`/deploy/jbossesb.esb/message-store-ds.xml`**.

Warning

Red Hat do NOT recommend the use of HSQLDB in production environments. It is not supported in that capacity. It has a number of limitations and is included only for use in testing and demonstrations.

The JBoss Enterprise SOA-P includes the *Database Schema Tool* for datasource configuration. The Database Schema Tool is an **`ant`** script located in ***`${SOA_ROOT}`*`/tools/schema/`**. You can run this script multiple times in order to change your configuration, so long as no additional changes have been made. If you do make additional changes, then the script may no longer work as expected.

To run the script, execute the command **`ant`** in that directory.

```
[ schema]$ ant
Buildfile: build.xml

[echo] JBoss SOA Platform
[echo]
[echo] Database Configuration Script
[echo]
[echo] This script is used to configure the SOA platform and all it's
[echo] constituent components against a new RDBMS. Deployment scripts
[echo] are currently available for the following databases: MySQL,
[echo] Oracle9i, PostgresSQL, Oracle10g
[echo]
[echo] NOTICE: THIS SCRIPT MAY NOT WORK CORRECTLY IF YOU HAVE MADE MANUAL
[echo] CHANGES TO THE DATABASE CONFIGURATION.
[echo]
```

The script will prompt you for the details it requires. These are as follows:

- The type of database you are using. Only MySQL, PostgreSQL, Oracle 9i and Oracle 10g are supported at this time.
- The name of the database instance.
- The hostname or Internet Protocol (IP) Address of the database.
- The Transmission Control Protocol (TCP) port being used for the database instance.
- The username that must be input in order to access the database.
- The password needed to gain access to the database.

Note

You can also specify these values in the file ***${SOA_ROOT}*/tools/schema/build.properties** before running the script.

Important

The Enterprise Service Bus Management Console must be re-deployed after the data-source configuration has occurred.

1.3.12. Using a *JSR-170 Message Store*

The JBoss Service-Oriented Architecture Platform allows one to utilise multiple message store implementations via a plug-in based architecture. One viable alternative to using the default database message store is to use a JSR-170 compliant *Java Content Repository* (JCR.)

The Java Content Repository implementation included is called *Apache Jackrabbit*. To enable the Java Content Repository's message store, add the following property to the "core" section of the **jbossesb-properties.xml** file, (which can be found in the root of the **jboss-esb.sar** or the root of **deployers/esb.deployer** on AS5):

```
<property name="org.jboss.soa.esb.persistence.base.plugin.jcr"
value="org.jboss.internal.soa.esb.persistence.format.jcr.JCRMessageStorePlu
gin"/>
```

By doing this, one adds the Java Content Repository plug-in to the list of available message stores. The Java Content Repository message store can use either an existing repository via JNDI or can generate a stand-alone instance locally on the Application Server.

In order to configure repository access, one should add the following list of properties to the "dbstore" section of the **jbossesb-properties.xml** file:

```
<property name="org.jboss.soa.esb.persistence.jcr.jndi.path"
  value="jcr"/>
<property name="org.jboss.soa.esb.persistence.jcr.username"
  value="username"/>
<property name="org.jboss.soa.esb.persistence.jcr.password"
```

```
  value="password"/>
<property name="org.jboss.soa.esb.persistence.jcr.root.node.path"
  value="JBossESB/MessageStore"/>
```

- jcr.jndi.path This is an optional path in JNDI to determine where the repository is found. If it is not specified, a new repository will be created based on the **`repository.xml`** file located in the root of **`jbossesb.sar`**. In this case, repository data is stored in the **`JBossAS/server/{servername}/data/repository`** directory.
- jcr.username - This is the user name needed to obtain a repository session.
- jcr.password - This is the password needed to access a repository session
- jcr.root.node.path - This is the path relative to the root of the repository in which messages shall be stored.

In order to quickly test that one's Java Content Repository message store is configured properly, one should add the **`org.jboss.soa.esb.actions.persistence.StoreJCRMessage`** action to an existing service. The action will then attempt to store the current message to the Java Content Repository, thereby letting one know if it works correctly.

1.3.13. *Message Tracing*

It is possible for one to trace any and all messages sent through the JBoss Service-Oriented Architecture Platform. This is may be something which one needs for any number of reasons, including the desire to see the audit trail or the need to debug a problematic process. Note that messages must be uniquely identified using the `MessageID` field of the Message header in order to be traced. This is referred to in more detail within the *Programmers' Guide*. This unique identification is the only way in which messages can be identified within the system.

The components of the JBoss Service-Oriented Architecture Platform (including gateways, the *ServiceInvoker* and the *load balancer*) are configured, by default, to log all of their interactions with messages. They do so by generating further log messages. These log messages will contain the header information associated with the message, thus enabling one to correlate them across multiple Service-Oriented Architecture Platform instances. These messages can be identified by looking for the following output:

```
header: [ To: EPR: PortReference < <wsa:Address ftp://foo.bar/> >,
From: null, ReplyTo: EPR: PortReference < <wsa:Address http://bar.
foo/> >, FaultTo: null, Action: urn:dowork, MessageID: urn:foo/bar
/1234, RelatesTo: null ]
```

Furthermore, one can enable a logging meta-data filter, the only role of which is to issue notes to the log whenever a message is either input to a component or output from it. This filter, **`org.jboss.internal.soa.esb.message.filter.TraceFilter`**, can be set, in conjunction with any others, within the "`Filter`" section of the **`JBossESB`** configuration file. Note that it has no effect upon the input or output message. Whenever a Message passes through this filter, you will observe the following log at the "information" level:

```
TraceFilter.onOutput ( header: [ To: EPR: PortReference < <wsa:Add
ress ftp://foo.bar/> >, From: null, ReplyTo: EPR: PortReference <
<wsa:Address http://bar.foo/> >, FaultTo: null, Action: urn:dowork
```

```
, MessageID: urn:foo/bar/1234, RelatesTo: null ] )
```

```
TraceFilter.onInput ( header: [ To: EPR: PortReference < <wsa:Addr
ess ftp://foo.bar/> >, From: null, ReplyTo: EPR: PortReference < <
wsa:Address http://bar.foo/> >, FaultTo: null, Action: urn:dowork,
MessageID: urn:foo/bar/1234, RelatesTo: null ] )
```

The *TraceFilter* will only log information if the property entitled org.jboss.soa.esb.messagetrace is set to **on/ON**. The default setting is **off/OFF**. If one enables, it all messages that pass through it shall be logged. However, one may desire finer-grained control over which messages are logged and which are ignored. In order to do this, one should make sure that the property org.jboss.soa.esb.permessagetrace is set to **on/ON**. Those Messages with a property of org.jboss.soa.esb.message.unloggable set to **yes/YES** shall now be ignored by this filter.

1.3.14. *Clustering and Fail-Over Support*

The JBoss Service-Oriented Architecture Platform has support for fail-over of "stateless" services. You should consult the *Programmers' Guide* for detailed information on this topic, but the pertinent issues to note are as follows:

- Because the ServiceInvoker hides much of the fail-over complexity from users, it only works with "native" SOA Platform messages. Furthermore, not all gateways have been modified to use the ServiceInvoker, so incoming messages that are unaware of the SOA Platform may not always be able to take advantage of service fail-over. The *Release Notes* document contains more details related to this topic.
- When the ServiceInvoker tries to deliver a message to it may now possibly be faced with a choice of multiple *End-Point References* (EPR). In order to help it determine which of these to select, one can configure a "*policy*." In the **jbossesb-properties.xml** file, a property entitled org.jboss.soa.esb.loadbalancer.policy can be set. At this point in time, three such policies are provided and, of course, one can create one's own.

  1. First Available: If a "healthy" *ServiceBinding* is found, it will be used until it dies, and then the policy will move to the next End-Point Reference in the list. This policy does not provide any load balancing between two service instances.
  2. Round Robin: This is a typical load balance policy whereby each End-Point Reference is "hit" in list-order.
  3. Random Robin: This policy is similar to that for the "Random Robin", the difference being that selection is randomized.

- The End-Point Reference list used by the Policy may well become smaller over time as "dead" references are removed. When the list is exhausted or the time-to-live of the list cache is exceeded, the ServiceInvoker will obtain a fresh list of End-Point References from the Registry. The org.jboss.soa.esb.registry.cache.life property defaults to 60000 milliseconds but this can be set to a different value in the **jbossesb-properties** file.
- If none of the End-Point References work, one should consider this a good situation in which to use the *Message Re-delivery Service*.
- If it is your intention to run the same service on more than one node in a cluster, you shall have to wait for service registry cache re-validation to take place. Until this time, the service will not be fully

working in the clustered environment. You can configure this cache re-validation time-out value in **`deploy/jbossesb.sar/jbossesb-properties.xml`** file.

```
<properties name="core">
  <property name="org.jboss.soa.esb.registry.cache.life" value="60000"/>

<!-- 60 seconds is the default -->
</properties>
```

- If one sets the org.jboss.soa.esb.failure.detect.removeDeadEPR property to "true," then whenever the ServiceInvoker "suspects" that an End-Point Reference has failed, it shall remove it from the Registry. The default setting is "false" because this should be used with extreme care. The End-Point Reference for a service that is simply overloaded and, therefore, slow to respond may, inadvertently, be removed from the Registry by mistake. These "orphaned" services will receive no further interactions and may have to be restarted.

1.3.15. Using OpenSSO in Conjunction with the SOA-P

The JBoss Service-Orientated Architecture Platform includes the *Open Web SSO* project (*OpenSSO*) software. This is used in order to simplify the implementation of a transparent *Single Sign-On* (SSO) service.

In order to obtain more information about the OpenSSO, please visit the project's website at: *http://opensso.dev.java.net*

1.3.15.1. Installing and configuring the OpenSSO in *Tomcat*

Note

There is an known issue with regard to deploying the OpenSSO on the JBoss Enterprise Service-Oriented Architecture Platform. However, the OpenSSO software can be deployed to other web-containers, which can then be used with the SOA-P.

If you desire to read the details of this deployment issue, they can be found at *https://jira.jboss.org/jira/browse/SOA-731*.

The following instructions are provided for deploying the OpenSSO to Tomcat. Information about using the OpenSSO with other web-containers can be found at *https://opensso.dev.java.net/public/use/docs/fampdf/index.html*.

Procedure 1.2. Deploying the OpenSSO onto Tomcat

1. Download the *Tomcat* software from the Apache project website at: *http://tomcat.apache.org*

Note

Red Hat customers can obtain and install Tomcat from the software repository, without needing to download it from the project's website.

2. Un-archive the file in a directory. (The following examples assume that this directory is called **/opt/tomcat**.)

3. Edit the **/opt/tomcat/bin/catalina.sh** shell script (or **catalina.bat** batch file if one is using a Windows deployments) and add *-Xmx1G* to the JAVA_OPTS property. This specifies the maximum heap size of the *Java Virtual Machine* instance is to be one gigabyte.

   Here are the configuration details that enable one to set the maximum size for JAVA_OPTS:

   ```
   JAVA_OPTS="$JAVA_OPTS "-Xmx1G" "-
   Djava.util.logging.manager=org.apache.juli.ClassLoaderLogManager"
   ```

   Example 1.2. Adding a Maximum Size Parameter to JAVA_OPTS

4. Now download **opensso.zip** (build 4.5) from the OpenSSO website: *https://opensso.dev.java.net/public/use/index.html*

5. Unpack the **opensso.zip** archive file and copy the **opensso.war** from the **deployable-war/** directory to **/opt/tomcat/webapps/**

6. If one wishes to deploy JBoss Enterprise Service-Oriented Architecture Platform and Tomcat on the same machine one can alter the Tomcat port that is specified in the **$tomcat/server.xml** as per this example:

   ```
   <Connector port="8090" protocol="HTTP/1.1">
   <Connector port="8099" protocol="AJP/1.3" redirectPort="8443"/>
   ```

   Example 1.3. Updating Tomcat port

7. Start Tomcat by running the **/opt/tomcat/bin/startup.sh** shell script (or the **startup.bat** batch file, if one is using a Microsoft Windows deployment.)

8. Open *http://localhost:8090/opensso* in a web browser client.

9. Click on **Create Default Configuration** with your mouse.

10. Enter **adminpass** for the **Default User[amAdmin]** and **ldappass** for **Default Agent [amldapuser]**

11. Click on **Create Configuration**. This will cause the OpenSSO to automatically configure itself.

12. Now, open *http://localhost:8090/opensso* again. Log in using the proper credentials. The user name one must input is **amAdmin** and the password is that which you chose to go with amAdmin.

One can find further information about installing the OpenSSO on Tomcat at this weblog entry: *http://blogs.sun.com/JohnD/entry/how_to_install_tomcat_6*.

1.3.15.2. Configuring the OpenSSO for Use in Conjunction with the JBoss SOA-P

The **AuthContext** class (found in the **openssoclientsdk.jar** Java archive file) performs the authentication process. The following steps describe the configuration one is required to undertake in order to enable this integration between the OpenSSO and the JBoss Service-Oriented Architecture Platform.

Procedure 1.3. Configuring the OpenSSO for Integration with JBoss SOA-P

1. Firstly, one must edit the **login-config.xml** file as illustrated below in order to provide the ability to integrate with OpenSSO.. This file is located in the server's **conf/** directory. The full file path may, for example, be **/jbossas/server/default/conf/login-config.xml**.

```
<application-policy name="OpenSSOLogin">
 <authentication>
   <login-module
       code="org.jboss.soa.security.opensso.OpenSSOLoginModule"
       flag="required">
     <module-option name="orgName">opensso</module-option>
     <module-option name="moduleName">DataStore</module-option>
     <module-option name="amPropertiesFile">
       props/AMConfig.properties
     </module-option>
   </login-module>
 </authentication>
</application-policy>
```

Example 1.4. Editing **login-config.xml** for the OpenSSO

The 'orgName' and the 'moduleName' are the same as those that you configured in the OpenSSO system. The last property indicates where on the system the **AMConfig.properties** file resides.

2. Next, the **AMConfig.properties** file must be edited. This file is located in the **conf/props/** subdirectory of your server. The full file path may, for example, be **/jbossas/server/default/conf/props/AMConfig.properties**.

   It has been configured, by default, to expect the server to be accessible on **localhost**, port **8080** with the **opensso** context path. If one wishes to change these to a customised configuration, or in order adopt a pre-existing, deployed OpenSSO, it is suggested that one uses the **scripts/setup.sh** shell script (or the **setup.bat** batch file on Microsoft Windows deployments) to undertake the changes to the settings.

   Next, run the shell script entitled **/samples/fam-client/sdk/scripts/setup.sh**. This can be found in can be found in the **opensso.zip** archive file. Once one has run it, one will see the following output onscreen:

```
Debug directory (make sure this directory exists): /var/local/tmp
Password of the server application: opensso1
Protocol of the server: http
Host name of the server: putian.nay.redhat.com
```

```
Port of the server: 8080
Server's deployment URI: opensso
Naming URL (hit enter to accept default value, http://
putian.nay.redhat.com:8080/opensso/namingservice):
```

Copy the **`AMConfig.properties`** from the **`$opensso.zip/samples/fam-client/sdk/resources/AMConfig.properties`** file.

For more information about the OpenSSO configuration, please read the documentation to be found at: *http://opensso.dev.java.net*.

Having successfully undertaken the two steps above, one is now able to use the OpenSSOLogin module as a JAAS plug-in provider.

Furthermore, one can use it as an identity provider, in order to secure the Service-Oriented Architecture Platform in this following way:

```
<service category="OpenSSO"
    name="SimpleListenerSecured" description="Hello World">
  <security moduleName="OpenSSOLogin" runAs="adminRole"/>
  <listeners>
    <jms-listener name="JMS-Gateway" busidref="quickstartGwChannel"
      maxThreads="1" is-gateway="true"/>
  </listeners>

  <actions mep="OneWay">
    <action name="debug" class="org.jboss.soa.esb.actions.SystemPrintln">
      <property name="printfull" value="false"/>
      <property name="message" value="In Service1"/>
    </action>
  </actions>
</service>
```

The Registry

At the heart of all JBoss Service-Oriented Architecture Platform deployments is the *Registry*. This is fully described in the *JBoss SOA Platform Services Guide*, in which configuration information is also discussed. However, it is worth noting the following:

- When services run, they typically place the *End Point Reference* (EPR), through which they can be contacted, within the registry. If they have been correctly developed, then services should automatically remove these End Point References from the registry when they terminate. However, in certain circumstance, entries can, potentially, be left within the registry. Some causes of these situations include machine crashes and incorrect programming. These "stale" entries prevent the correct execution of subsequent deployments. If this occurs, these entries can be removed manually. However, it is obviously important that one ensures that the system is in an inactive state before doing so.
- If one sets the optional `remove-old-service` tag name in the End Point Reference to "`true`", then the ESB will remove any existing service entry from the Registry prior to adding a new instance. However, this option should be used with care, because the entire service will be removed, including all End Point References.

Updated. Configuring Web Service Integration

The JBoss Service-Oriented Architecture Platform exposes *Web Service Endpoints* through the *SOAPProcessor* action. This action integrates the JBoss Webservices v2.x container into the Service-Oriented Architecture, allowing one to invoke *JBossWS* Endpoints over any channel that is supported by the SOA-P.

Note that one is required to have JBossWS 2.0.1.SP2 (native) or higher properly installed on one's JBoss Service-Oriented Architecture Platform server in order to use the the SOAPProcessor action.

You should refer to the *Programmers' Guide* for more details.

Default "ReplyTo" Endpoint References

The JBoss Service-Oriented Architecture Platform employs *Endpoint References* (EPRs) to address messages going to and from services. As described in the *Programmers' Guide*, messages have headers that contain recipient addresses and sequential numbers (for the purpose of correlation.) They may also contain further optional addresses for replies, faults and so forth. Because the recommended interaction pattern within the JBoss Service-Oriented Architecture Platform is based on a one-way message exchange, responses to messages do not necessarily occur automatically; it is dependent on the individual application as to whether or not a sender expects a response.

A reply address is an optional part of the header's routing information, which an application can set if necessary. When a response is required and the ReplyTo End-Point Reference has not been set, the JBoss SOA-P will use a default value specific to each type of transport. Some of these "ReplyTo" defaults require system administrators to perform additional configuration of the JBoss Service-Oriented Architecture Platform in order to work correctly:

- If the Java Messaging Service is being used, the software assumes it to be a queue with the same name as that which was used to deliver the original request, which was prefixed with '_reply.'
- If it is the JDBC that is being employed, the software assumes that it uses a table in the same database with the same name as that which was used to deliver the original request, which was prefixed with '_reply_table.' Also, one must bear in mind that the new table needs the same columns as the request table.
- If local and remote files are being used, then no administration changes are required. Responses are written into the same directory as the request, albeit with a unique suffix in order to ensure that only the original sender will collect the response.

The ServiceBinding Manager

If you wish to run multiple JBoss Service-Oriented Architecture Platform servers on the same machine, you may want to utilise the functionality of the JBoss *ServiceBinding Manager*. This program allows you to centralise the port configuration of all of the instances that you will be running. The SOA-P server ships with a sample bindings file, named **`docs/examples/binding-manager/sample-bindings.xml`**.

The JBoss Application Server documentation contains detailed instructions with regard to how to configure the ServiceBinding Manager.

Note

If you are using `jboss-messaging` as your Java Messaging Service provider, your ServiceBinding Manager configuration for it must match the contents of the **`remoting-service.xml`** file.

Monitoring and Management in the ESB

A number of options are available with regard to the need to monitor and manage the *Enterprise Service Bus* server. These include tools that are shipped with the ESB itself, such as various useful JMX *MBeans* that can be utilised in order to analyse performance.

In the **`jboss.esb`** domain, one should see the following types of MBeans:

- deployment=<ESB package name> – This MBean shows the state of all of the Enterprise Service Bus packages that have been deployed. It is also able to give information about their XML configurations and their current statuses.
- listener-name=<Listener name> – This MBean displays all deployed "listeners," along with such information as their XML configuration, start time, `maxThreads` and state. The administrator is given the options of initializing, starting, stopping or destroying a listener.
- category=MessageCounter – This MBean displays all of the services deployed for a "listener," the separate actions of each aforementioned service and the count of how many messages were processed, as well as the accompanying processing time of each message.
- service=<Service-name> – This last MBean displays a variety of statistics for each service including message counts, states, average sizes and processing times. The message counts may be reset and services may be stopped and started via this MBean.

In additional, the Java Message Service-domain MBeans show statistics for message queues, which one might find quite useful for the purpose of debugging or determining performance issues.

6.1. *Updated* Monitoring and Management in the SOA-P

The JBoss Service-Oriented Architecture Platform provides management and monitoring functionality via the embedded *JOPR* interface. *http://localhost:8080/admin-console*

The JBoss Service-Oriented Architecture Platform's *monitoring console* is designed to gather information on the performance of different deployed services. As of JBoss Enterprise Service Bus Release 4.2.0.GA, the monitoring console grants users the ability to obtain message counts by service, action and node. It also provides other information like processing time, number of failed messages, bytes transferred and dates and times for "last successful" and "failed" messages. Please note that, as of JBoss Enterprise Service Bus Release 4.6, the previous ESB monitoring tool has been deprecated.

The monitoring console is automatically installed in both the JBoss Application Server and the stand-alone Enterprise Service Bus server.

Below, one can see a screen-shot of the console. It is programmed to request and display MBean information from each of the nodes contained within the Enterprise Service Bus registry.

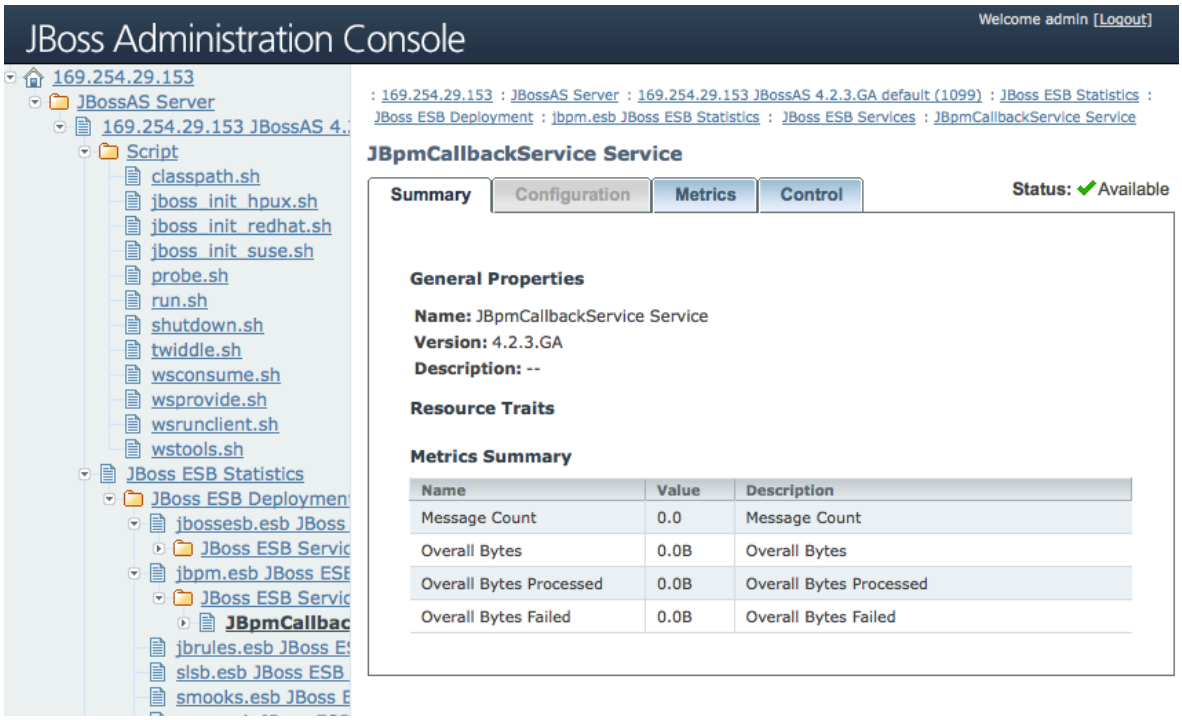

Figure 6.1. JBoss Administration Console

6.1.1. Services

Each ESB service is displayed along with the processing time per action, processed count per action, failed count per action and overall message count (per service), as can be witnessed in the following screen-shot:

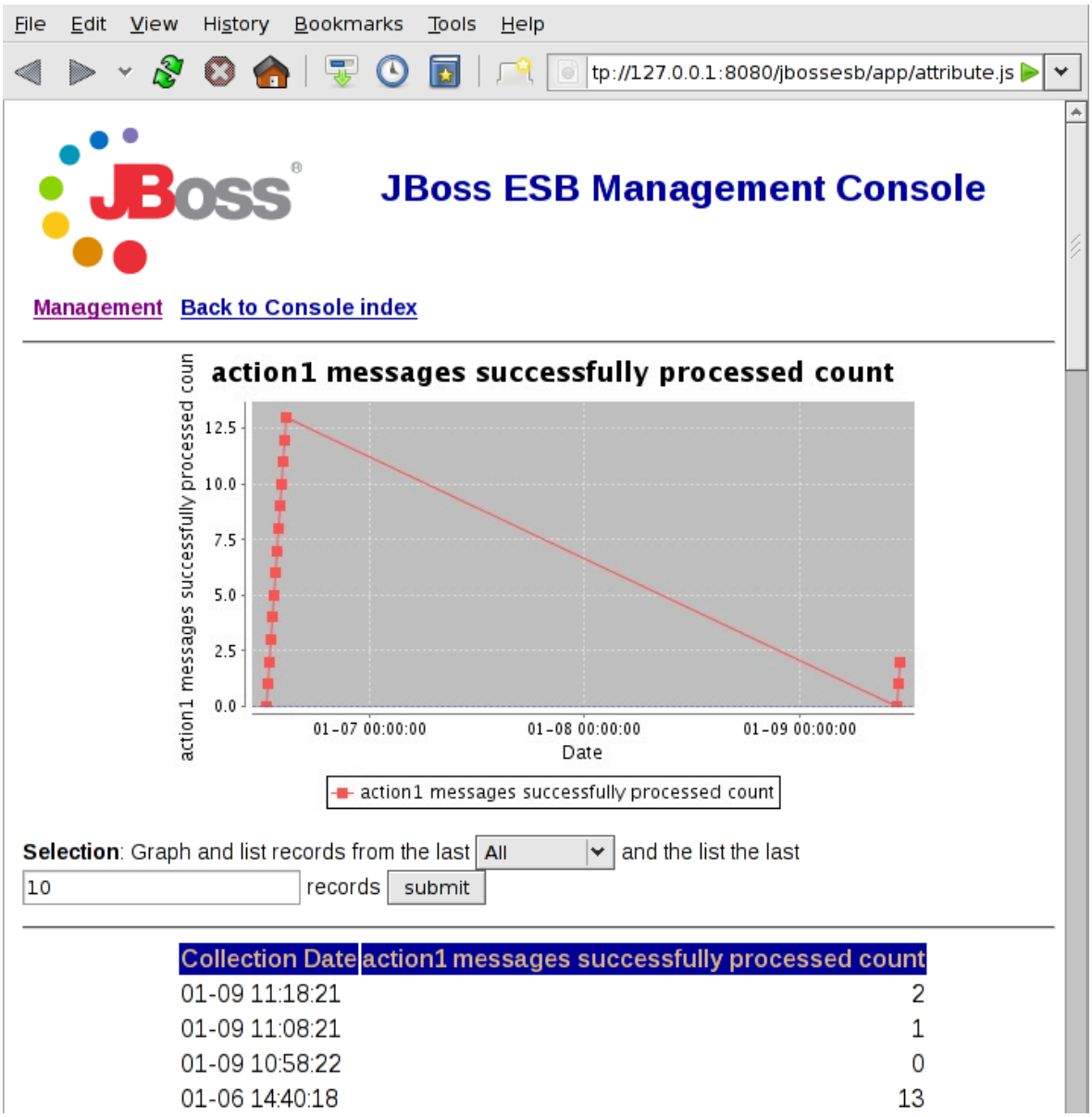

Figure 6.2. Services

6.1.2. Message Counter

The monitoring console also provides one with an overall count of the messages that pass through the Enterprise Service Bus. The *Message Counter* keeps track of the number of both successful and failed messages, along with their processing times and the dates.

6.1.3. Smooks Transformations

There is an MBean that keeps track of the count of "processed" *Smooks Transformations* and the time taken to process each. It also keeps track of the overall count of the *Transformation Chain*.

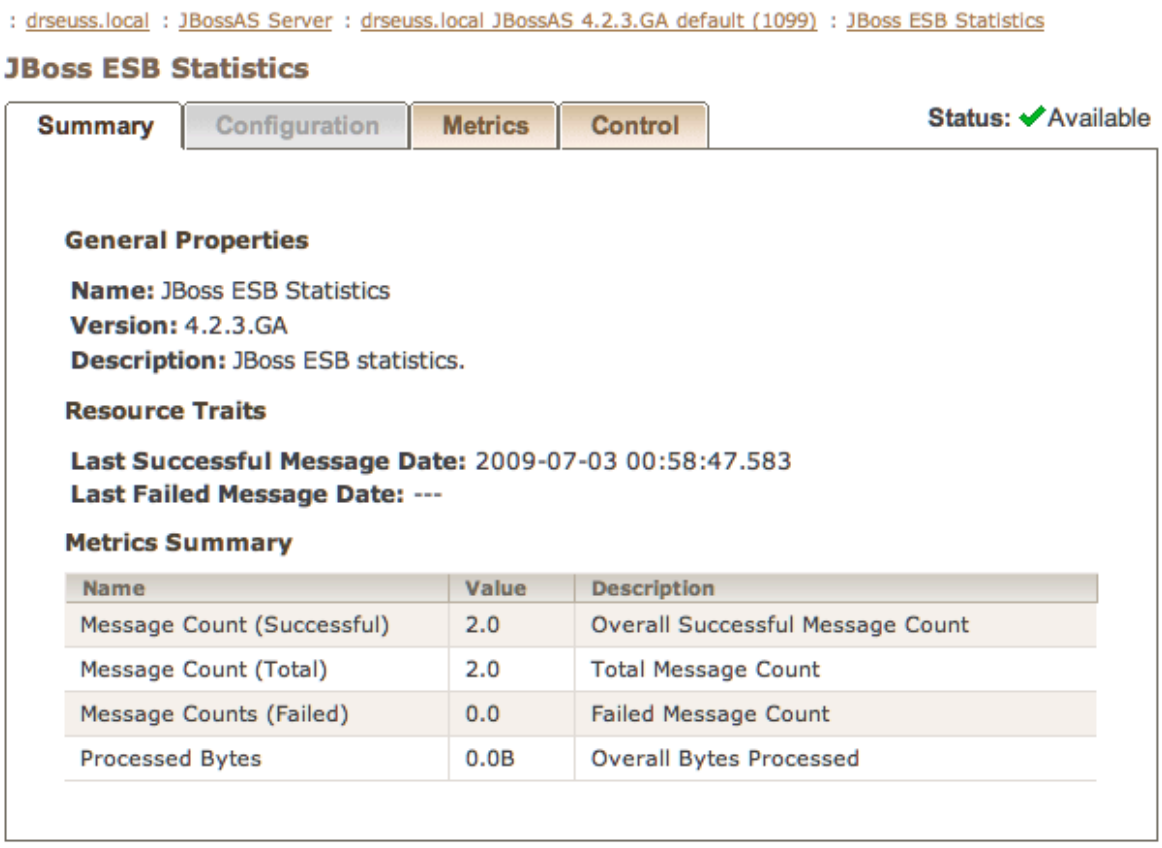

Figure 6.3. MBean Counter

6.1.4. Dead Letter Service

As has been mentioned in the *Programmers' Guide*, the *DeadLetterService* (DLQ) can be used to store those messages that cannot be delivered. This is a JBoss ESB service and can, therefore, be both monitored and inspected. Note, however, that the `DeadLetterService` is not used if the underlying transport has native support, as, for example, is the case with the Java Messaging Service. In that case, you should inspect the JBoss Enterprise Service Bus Dead Letter Service as well as any transport-specific equivalent.

6.2. Alerts

The JBoss *Web Console* *https://wiki.jboss.org/auth/wiki/WebConsole* is an utility program that is available within both the JBoss Application Server and the JBoss Enterprise Service Bus Server. It is capable of monitoring and sending alerts based on their JMX MBean properties. One can use this functionality to receive alerts for ESB-related events, such as, for instance, when the `DeadLetterService` counter reaches a pre-determined threshold.

1. One must, firstly, configure the *./deploy/mail-service.xml* file with one's own *Simple Mail Transfer Protocol* (SMTP) settings.

2. Next, modify the **`./deploy/monitoring-service.xml`** file by removing the comment markers from the `EmailAlertListener` section. After doing so, add the appropriate header-related information.

3. Finally, create a file named **`./deploy`**. (This will serve as the monitoring **`MBean`**.) It should contain the following code:

```
<?xml version="1.0" encoding="UTF-8"?>
<server>
 <mbean code="org.jboss.monitor.ThresholdMonitor"
  name="jboss.monitor:service=ESBDLQMonitor">
  <attribute name="MonitorName">
    ESB DeadLetterQueue Monitor
  </attribute>

  <attribute name="ObservedObject">
   jboss.esb:category=MessageCounter,
deployment=jbossesb.esb,service-name=DeadLetterService
  </attribute>

  <attribute name="ObservedAttribute">
    overall service message count
  </attribute>
  <attribute name="Threshold">4</attribute>
  <attribute name="CompareTo">-1</attribute>
  <attribute name="Period">1000</attribute>
  <attribute name="Enabled">true</attribute>
  <depends-list optional-attribute-name="AlertListeners">
    <depends-list-element>
      jboss.alerts:service=ConsoleAlertListener
    </depends-list-element>
    <depends-list-element>
      jboss.alerts:service=EmailAlertListener
    </depends-list-element>
  </depends-list>
  <depends>jboss.esb:deployment=jbossesb.esb</depends>
 </mbean>
</server>
```

Having done so, you will find that once the `DeadLetterService` counter reaches "five," it will send an e.-mail to the address(es) specified in the **`monitoring-service.xml`** file. Note that the alert is sent only once, this being when that threshold is reached. If you desire to be alerted again once the counter is re-started, you can also reset the alerted flag on your monitoring service MBean (in this case, called `jboss.monitor:service=ESBDLQMonitor`.)

For more information and advice on how to use the JBoss Web Console monitoring, please read *http://wiki.jboss.org/auth/wiki/JBossMonitoring*.

6.3. JON for SOA

An additional option that you have for monitoring and administering your JBoss SOA-P server is the *JBoss Operations Network* (JON) product.

The JBoss Operations Network software provides functionality for inventorying, administration, monitoring, deployment and updating within the context of JBoss-based middleware applications.

It performs these tasks via a centrally-managed model with a customizable web-portal interface. Additional details on this product can be found on the project website at *http://www.jboss.com/products/jbosson*.

"*JON for SOA*" is a stand-alone release of JON that includes functionality that has been specifically designed for the JBoss SOA-P. This section provides an overview of that functionality and assumes a basic knowledge of the JBoss Operations Network product on behalf of the readership.

Important

Contrary to the case with regard to embedded JBoss SOA Platform consoles, access to JON is not restricted to the local server. This grants one greater freedom in its use but also means that one cannot rely on those inherent restrictions to ensure the security of the console.

Adding your JBoss SOA Platform Server to the JON Inventory

Your JBoss SOA Platform server will appear in JON as a resource of the type "JBossAS Server". It will have the description of "JBoss Enterprise SOA Platform".

When you first try to access your SOA-P server in the JBoss Operations Network console, you will see an error message. This is because you have not yet provided the authentication details required of a valid SOA-P user.

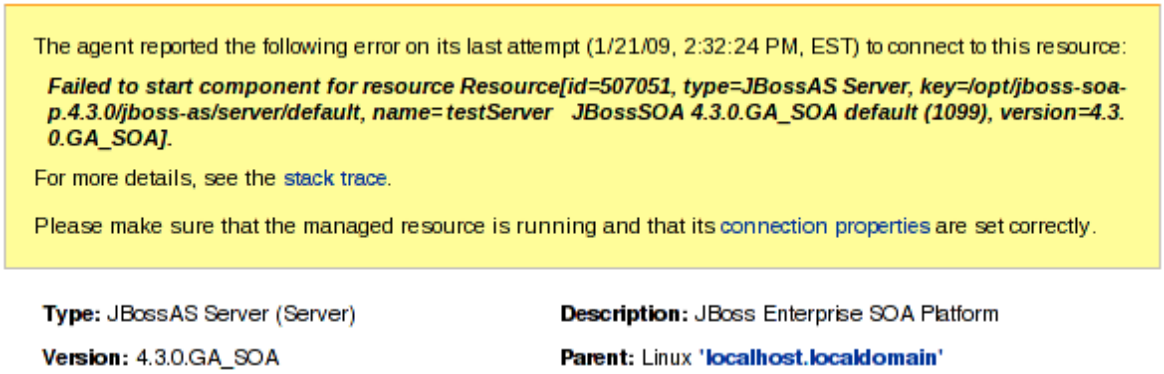

Figure 6.4. The error displayed when authentication information required is incorrect or absent.

The user information will be found within that **`conf/props/soa-users.properties`** file associated with the server profile in use. One enters this information as the Principal and Credentials (user name and password) in the server's `Connection Properties`. One accesses these details by selecting the **Server** and then the **INVENTORY** tab. Conveniently, the error message also contains a short-cut link to the "Connection Properties" page.

JBoss SOA-P Enterprise Service Bus Statistics

Once one's JBoss SOA-P server is correctly configured for the JBoss Operations Network tool, one should find the item **JBoss ESB Statistics** now available underneath the menu entry in **Resources** on the **MONITOR** tab.

Clicking on **JBoss ESB Statistics** enables one to "drill down" into the Enterprise Service Bus statistics. At this level, the figures displayed are those for an overall summary of the ESB instance.

If one clicks on the **JBoss ESB Deployment** item, a list of all the Enterprise Service Bus packages deployed on the server will be displayed. Note that no statistics are displayed at this level.

By selecting an ESB deployment, one can drill down into it and observe its statistics. From here, one can also drill down further to view the details for the services and actions that make up the deployment.

The metrics collected and displayed vary, depending on the Enterprise Service Bus component in question. The available metrics include:

- Message Count
- Message Count (avg)
- Messages Failed
- Messages Failed (avg)
- Messages Successfully Processed
- Messages Successfully Processed (avg)
- Overall Bytes
- Overall Bytes Failed
- Overall Bytes Processed
- Processing Time
- Message Count
- Message Count (avg)
- Deployment Type
- .esb State
- .esb State String
- Message Counts (Failed)
- Message Counts (Successful)
- Message Counts (Total)
- Processed Bytes
- Last Failed Message Date
- Last Successful Message Date
- State
- Lifecycle State
- Maximum Number of Threads

- MEP
- Service Category
- Service Description
- Service Name
- Start Date

Based on these statistics, all the standard JBoss Operations Network functionality including alerts and charts can be configured for an ESB Deployment, Service or Action.

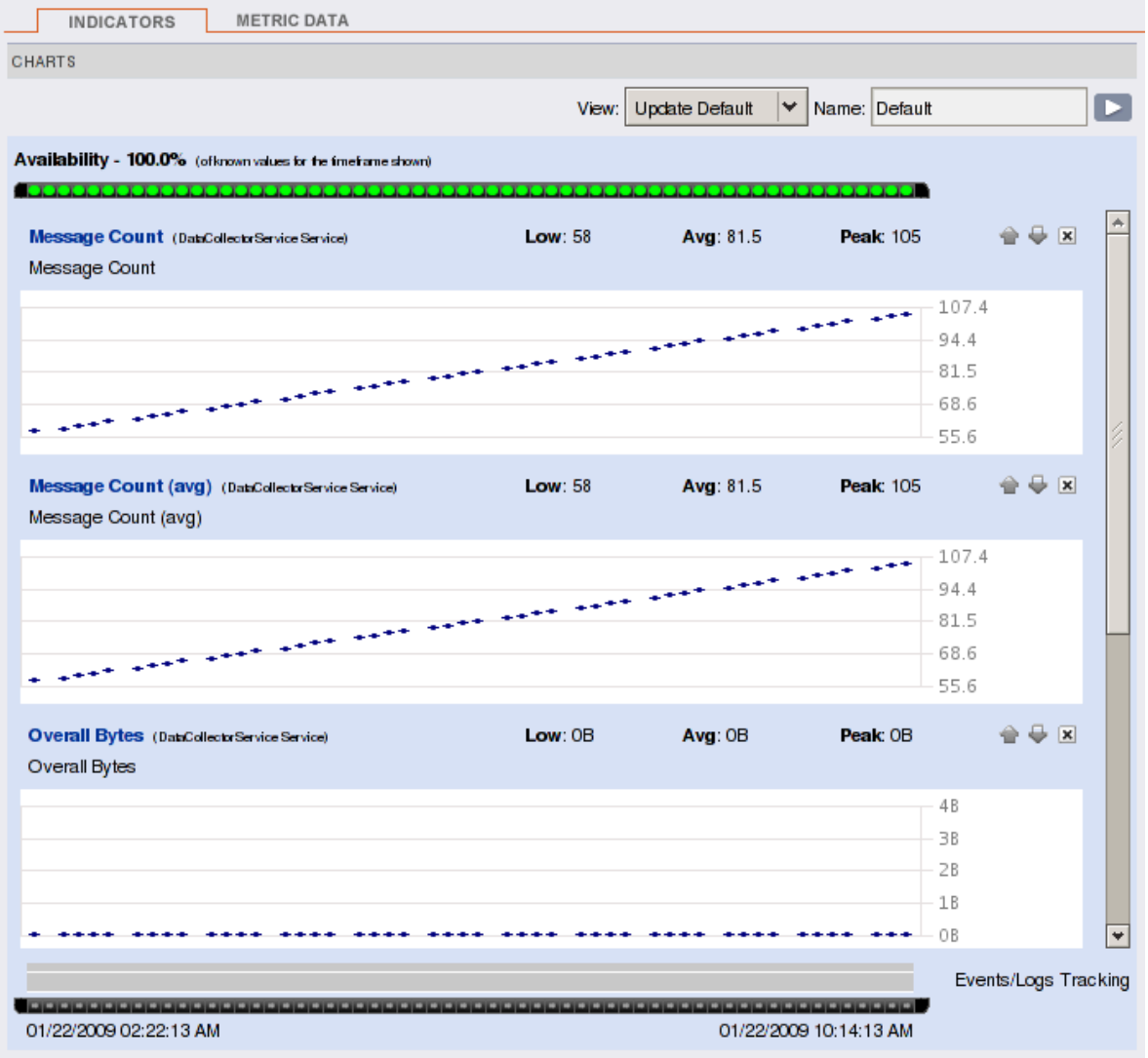

Figure 6.5. Metrics Displayed

Managing deployed ESB archives

JON for SOA also includes functionality for deploying or deleting the Enterprise Service Bus archives on your server. This functionality is found in the **Child Resources** section of the **INVENTORY** tab of the **`JBoss ESB Statistics`** screen.

A new Enterprise Service Bus archive can be deployed by selecting **JBoss ESB Deployment** from the **Create New** menu. Then, from the **Create New Resource** page, specify which archive to deploy, to where it should be sent (which, under normal circumstance, would be one's **`deploy`** directory) and, finally, whether it should be sent as a compressed or extracted archive.

Existing archives can be deleted. To do so, simply tick their entries as found in the **Child Resources** list and then click **DELETE**.

The history of previous Enterprise Service Bus deploy and delete requests can be viewed in this section as well.

Automatic Service Discovery

The *JBoss Operations Network Agent* will automatically detect ESB archives that have been deployed or deleted independently of the JON interface. Newly-deployed ESB archives are added to the server inventory automatically but deleted archives are not removed in the same way.

The default agent is configured to only perform this "service discovery" once every 24 hours. You have two means with which to change this time period:

1. Edit the **`conf/agent-configuration.xml`** file.

   This time period can be changed in the agent configuration file, (**`conf/agent-configuration.xml`**.) You have to restart the agent for this to take effect.

   ```
   <entry key="rhq.agent.plugins.service-discovery.period-secs"
    value="86400"/>
   ```

   Figure 6.6. Service discovery period setting in the **`conf/agent-configuration.xml`** file

2. Use the JON console to edit the configuration.

   JBoss Operations Network Agents can be added to your inventory of server resources. Once added, their configurations can be edited like those of any other inventoried resource. You can edit the **Service Discovery Period** value under the **CONFIGURE** tab of the applicable resource. This does not require a restart of the Agent in order to take effect.

In contrast to the SOA-P's embedded console, there is no way to force the JBoss Operations Network Console to perform an immediate collection of new data. Buttons such as **Get Current Values** in the **Metric Data** tab only update the display to reflect the most recently collected data. To obtain an immediate update, one can, however, set the collection period to a very low value such as thirty seconds (and then set the interval back to the previous figure afterwards.) For performance reasons, one is not recommended to lower the collection period by a significant amount.

Hot Deployment

7.1. Server Mode

The JBoss Service-Oriented Architecture Platform supports "*hot deployment*." The server regularly checks the **`deploy`** directory for new files. In addition, it also checks those files that have already been deployed for specific changes. When these changes are detected, the files are removed from deployment by the server and the new files are substituted in their place. This is referred to as "*hot re-deployment*."

The specific changes monitored vary by package type.

1. SAR files

   The **`jbossesb.sar`** is hot deploy-able. It will re-deploy when:

   - The archive's time-stamp changes, if the **SAR** file is compressed.
   - The timestamp of the **`META-INF/jboss-service.xml`** changes, if the **SAR** file is in exploded form.

2. **ESB** files

   Any **`*.esb`** archive will re-deploy when

   - The time-stamp of the archive changes, if the **ESB** file is compressed.
   - The time-stamp of the **`META-INF/jboss-esb.xml`** changes, if the **ESB** is in exploded form.

   The actions have "life-cycle support." This means that, upon hot re-deployment, they terminate "gracefully," finishing active requests. They will not accept any more incoming messages until they have re-started. (All of this occurs automatically.) If you wish to update just one action, you can use "*Groovy*" scripting to modify an action at run-time (see the "Groovy" Quick Start at *http://wiki.jboss.org/wiki/Wiki.jsp?page=JBossESBQuickStart*).

3. Rule Files

   There are two options available to you if you wish to refresh rule files (**`DRL`** or **`DSL`** files):

   - Firstly, you can re-deploy the **`jbrules.esb`** archive.
   - Alternatively, you can turn on the `ruleReload` feature in the Action Configuration (see *JBossESBContentBasedRouting* at *http://wiki.jboss.org/wiki/Wiki.jsp?page=JBossESBContentBasedRouting*). After doing so, if it is detected that a rule file has changed, it will be re-loaded.

4. Transformation Files

   There are two options you can pursue if you wish to refresh transformation files:

   - Firstly, you can re-deploy the **ESB** archive in which the transformation file resides.
   - Alternatively, you can send out a notification message over the JMS (topic) using the ESB console. The Smooks processors will receive this event and be prompted to re-load.

5. Business Process Definitions

   New versions of the *JBoss Business Process Management* tool's (jBPM) *Business Process Definitions* can be deployed to the jBPM database via the *Eclipse* plug-in. When you deploy a new version, it will only be used by fresh process instances. The life-cycles of existing processes will finish still using the previous definition. For more details, please refer to the jBPM documentation.

7.2. Stand-Alone ("Bootstrap") Mode

Enterprise Service Bus archives are not deployed when the *Boot-Strapper* mode is run . You can only have one **`jboss-esb.xml`** configuration file per node. The node will monitor the time-stamp on this file and will re-read it's configuration if a change occurs. In order to updates rules, one will have to use the '`ruleReload`' functionality.

Finally, to update Business Process Definitions, one can follow the same process outlined above.

Contract Publishing

If one wishes to integrate certain JBoss Service-Oriented Architecture Platform *endpoints*, one will possibly be required to supply information about that endpoint and the operations that it supports. This is particularly the case for *Web Service End Points* exposed via the *SOAPProcessor* action (see the *JBoss SOA Message Action Guide* for more detail on this topic.)

8.1. The "Contract" Application

For this purpose, **`Red Hat`** bundle the *Contract Application* with the JBoss SOA-P[1] . This application is installed by default along with the Enterprise Service Bus (after running "`ant deploy`" from the **`install`** directory.) [2]

It can be accessed via *http://localhost:8080/contract/*.

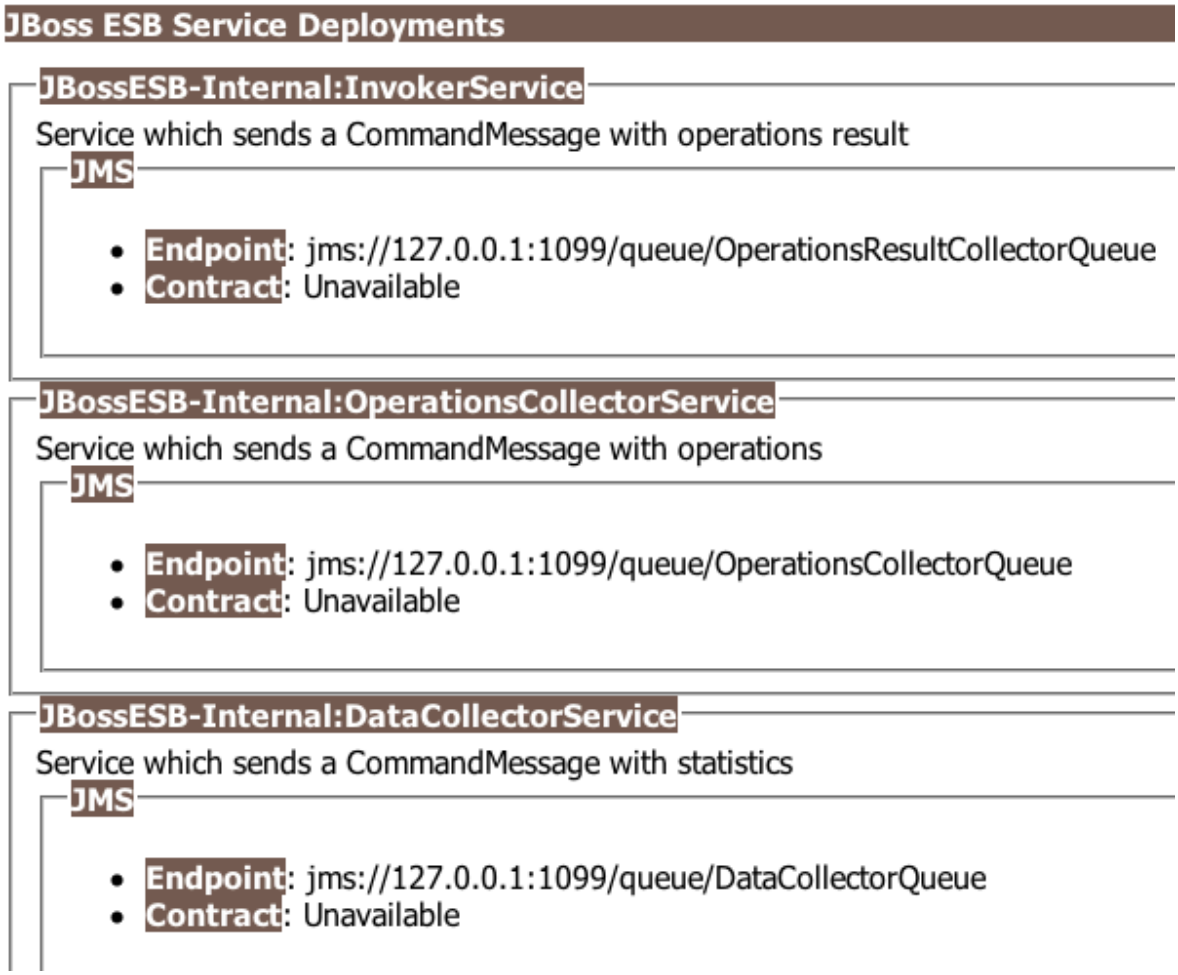

Figure 8.1. JBoss ESB Contract Application

As one can see, the application groups the endpoints according to the service with which they are associated (servicing). Another thing you may well notice is that some of the endpoints have an active

[1] This application is only being offered as a *Technical Preview* at this point in time. It will be superseded in a later release.
[2] Note that the Contract Application is also bundled inside the JBoss SOA Console. If you are deploying the console, you will first need to "un-deploy" the default Contract application. To do this, simply remove the **`contract.war`** web archive file from the **`default/deploy`** directory of your JBoss SOA-P Server.

"Contract" hyperlink. The ones that are visible in this case are for those Web Service End Points exposed via the `SOAPProcessor`. The hyperlink connects via the WSDL.

8.2. Publishing a Contract from an Action

The JBoss Enterprise Service Bus "discovers" endpoint contracts via the action pipeline that is configured on the Service. It initially looks for the first action in the pipeline that publishes contract information. If none of the actions do so, then the Contract Application displays the words "Unavailable on Contract" for that endpoint.

In order to publish contract information, an action receives the **`org.jboss.internal.soa.esb.publish.Publish`** annotation as follows. (This example uses the `SOAPProcessor` as for demonstrative purposes):

```
@Publish(JBossWSWebserviceContractPublisher.class)
public class SOAPProcessor extends AbstractActionPipelineProcessor {
    //TODO: implement
}
```

Some example `SOAPProcessor` source code can be found here: *http://anonsvn.labs.jboss.com/labs/jbossesb/trunk/product/services/soap/src/main/java/org/jboss/soa/esb/actions/soap/SOAPProcessor.java*[3]

One then needs to implement a "*ContractPublisher*" (`org.jboss.soa.esb.actions.soap.ContractPublisher`). This will require implementation of just a single method:

```
public ContractInfo getContractInfo(EPR epr);
```

Some example `JBossWSWebserviceContractPublisher` source code can be studied here: *http://anonsvn.jboss.org/repos/labs/jbossesb/trunk/product/services/soap/src/main/java/org/jboss/soa/esb/actions/soap/JBossWSWebserviceContractPublisher.java*

[3] http://jboss.org/repos/labs/jbossesb/trunk/product/services/soap/src/main/java/org/jboss/soa/esb/actions/soap/SOAPProcessor.java

Updated. JBoss Business Process Manager (jBPM)

9.1. jBPM Console

The *jBPM Web Console* is deployed by default as part of the **jbpm.esb** file and can be found at *http://localhost:8080/jbpm-console/*. Please refer to the jBPM documentation for detailed information regarding the Console.

9.2. jBPM Message and Scheduler Service

The JBoss Business Process Manager integration within the Enterprise Service Bus facilitates the support of additional messaging and scheduler services.

```
<service name="message"
    factory="org.jbpm.msg.db.DbMessageServiceFactory" />
<service name="scheduler"
    factory="org.jbpm.scheduler.db.DbSchedulerServiceFactory" />
<bean name="jbpm.job.executor" class="org.jbpm.job.executor.JobExecutor">
    ...
</bean>
```

These are distinct from those offered natively by jBPM. In addition to the standard jBPM configurations, Red Hat now also supports both a JMS-based message service, which is based on a JCA in-flow and a scheduling service based on JBoss Messaging.

To use the JMS-based Message Service and the JBoss Messaging-based Scheduler Service you must replace the default JBoss Business Process Manager configuration files in ***${SOA_ROOT}*/server/production/deploy/jbpm.esb/** with those provided in ***${SOA_ROOT}*/server/production/deploy/jbpm.esb/config/jmsscheduler/**.

```
$ cd ${SOA_ROOT}/server/production/deploy/jbpm.esb/
$ cp -fb config/jmsscheduler/jbpm.cfg.xml.config jbpm.cfg.xml
$ cp -fb config/jmsscheduler/jbpm-service.xml.config jbpm-service.xml
$ cp -fb config/jmsscheduler/jbm-queue-service.xml.config jbm-queue-
service.xml
```

Example 9.1. Replacing the jBPM configuration using the command-line on Linux

Note

Note that the names of the replacement configuration files have **.config** appended to them. You must, therefore, rename these.

Performance Tuning

10.1. *Updated.* Overview

There are various ways of tuning and optimising the performance of the JBoss Enterprise Service Bus for one's specific environment. Before doing this, however, you should realize that, as with any system, there is always a compromise between performance and reliability. The default configuration is designed for maximum reliability and stability, which may have an adverse affect on performance in certain circumstances.

10.2. *Updated.* `InVM Transport`

The term *InVM Transport* relates to functionality that means that a service can be invoked using the **`ServiceInvoker`** from within the same virtual machine with minimal impact upon resources. This is because it does not incur any networking or message serialisation overhead.

Important

Due to the volatility of the InVM queue, you may not be able to achieve all of the ACID semantics, particularly when this functionality is used in conjuction with other transactional resources, such as databases.

For additional reading on this topic, please refer to the "InVM Transport" section of the *Programmers' Guide*.

This code shows one how to configure one's service using the InVM Transport functionality:

```
<service category="HelloWorld" name="Service1"
                        description="Service 1" invmScope="GLOBAL">
    <listeners>
        <!-- So we just need to define a Gateway to the service... -->
        <jms-listener name="JMS-Gateway" busidref="quickstartGwChannel"
            is-gateway="true"/>
    </listeners>

    <actions>
        <action name="println"
            class="org.jboss.soa.esb.actions.SystemPrintln">
            <property name="message" value=" - > Service 1"/>
        </action>
        <!-- Route to the "Service 2" -->
        <action name="routeAction"
            class="org.jboss.soa.esb.actions.StaticRouter">
            <property name="destinations">
                <route-to service-category="HelloWorld"
                    service-name="Service2"/>
            </property>
        </action>
    </actions>
```

```
</service>
```

10.3. *Updated* Maximum Threads for the `MessageAwareListener`

The default value for the maximum number of threads allowed by **`MessageAwareListener`** is "one." This example source code shows how to change that value to a hundred:

```
<services>
    <service category="MyServiceCategory" name="MyWSProducerService1"
        description="WS Frontend speaks natively to the ESB"
        invmScope="GLOBAL">

        <property name="maxThreads">100</property>

        <listeners>
            <jbr-listener name="Http-Gateway" busidref="Http-1" is-
gateway="true" maxThreads="1"/>
        </listeners>

        <actions>
            <action name="println"
 class="org.jboss.soa.esb.actions.SystemPrintln">
                <property name="message" value=" - > Service 1"/>
            </action>
        </actions>
    </service>
</services>
```

10.4. *Updated* Maximum Threads for `jbr-listener`

The default value for the maximum number of threads allowed by **`jbr-listener`** is "fifty." This sample source code shows how to change that value to one hundred:

```
<services>
    <service category="MyServiceCategory" name="MyWSProducerService1"
        description="WS Frontend speaks natively to the ESB"
            invmScope="GLOBAL">
        <listeners>
            <jbr-listener name="Http-Gateway" busidref="Http-1"
                is-gateway="true">
                <property name="jbr-maxThreads" value="100"/>
            </jbr-listener>
        </listeners>
        <actions>
            <action name="println"
                class="org.jboss.soa.esb.actions.SystemPrintln">
```

```
                <property name="message" value=" - > Service 1"/>
            </action>
        </actions>
    </service>
</services>
```

10.5. *Update* Message Filters

Message filters are used to dynamically augment messages. For example, they can be used to add transaction or security information to a message when it flows through the Enterprise Service Bus. Using these filters may impact upon your system's performance. This is dependent upon the particular message filters that have been configured in your ESB. For further information on this topic, please refer to the "Meta-Data and Filters" section in the *Programmers' Guide*.

For more details on this overall subject, please refer to the "InVM Transport" section in the *Programmers' Guide*.

Appendix A. Revision History

Revision 3.0 Fri Oct 9 2009 David Le Sage *dlesage@redhat.com*

Update for Version 5.0

Revision 1.2 Wed Sep 1 2009 Darrin Mison *dmison@redhat.com*

Updated for SOA 4.3.CP02
SOA-1279 - Updated database details to refer to Schema Tool. Section 1.4
SOA-1310 - Added Performance Tuning Chapter. Chapter 10
SOA-1032 - Updated jBPM Chapter with alternative configuration details. Section 9.1
SOA-1261 - Updated IBM WebSphere MQ JCA Adapter details. Section 1.3.4.2

Revision 1.1 Tue Feb 24 2009 Darrin Mison *dmison@redhat.com*

Updated for SOA 4.3.CP01
Added Clustered ESB Service Configuration
Updated Database Schema Tool instructions
Updated OpenSSO installation
Updated ESB Console installation instructions

Revision 1.0 Fri Sep 5 2008 Darrin Mison *dmison@redhat.com*

Initial Creation