JBoss Enterprise SOA Platform 4.3

Administration Guide

Your guide to administering the JBoss Enterprise SOA Platform

JBoss Enterprise SOA Platform 4.3 Administration Guide
Your guide to administering the JBoss Enterprise SOA Platform Edition 1.0

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

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

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

The Administration Guide contains important information on how to configure and manage installations of JBoss SOA Platform 4.3GA.

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_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. Standalone server

If you wish to run the JBoss Enterprise SOA Platform ESB server on the same machine as JBoss Application Server (JBossAS), then you should look at *http://wiki.jboss.org/wiki/ConfiguringMultipleJBossInstancesOnOneMachine*.

1.2. JMS Providers

The JBoss Enterprise SOA Platform currently supports JBoss Messaging, IBM Websphere MQ Series (version 5.3 and 6.0) and Tibco EMS as Java Message Service (JMS) providers.

We recommend JBoss Messaging and it is included with the default configuration.

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

Important

This section is not intended as a replacement for the configuration documentation that comes with the supported JMS 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 a Queue or Topic by specifying the following parameters in their configuration (**`jbossesb-listener.xml`** and **`jbossesb-gateway.xml`**):

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

You will need to ensure that the client jar files of the JMS-provider you want to use are included in your classpath.

Each JMSListener and JMSGateway can be configured to use it's own JMS provider so you can use more than one provider in your deployment.

The SOA Platform utilizes a connection pool to improve performance when using JMS.
By default the size of this pool is set to 20, but this can be overridden by setting the

org.jboss.soa.esb.jms.connectionPool property in the transports section of the ESB configuration file. The service will keep retrying for up to 30 seconds if an initial session cannot be obtained. This time-out period can be configured using the org.jboss.soa.esb.jms.sessionSleep property.

Important

In the following sections we will make the following assumptions:

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

1.2.1. JBossMessaging

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

For JBossMessaging you should set the parameters to:

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

The jar file **`jboss-messaging-client.jar`** must be included in your classpath.

Configuring JBoss Messaging in a clustered setup gives you load balancing and failover facilities for JMS. Since this capability has changed between different versions of JBoss Messaging and may continue to do so, you should consult the JBoss Messaging documentation.

1.2.2. Apache ActiveMQ

Warning

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

For Apache ActiveMQ you should set the parameters 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 jars can be found in **lib/ext/jms/activemq**.

Apache ActiveMQ has been tested with the 4.1.0-incubator version.

1.2.3. IBM Websphere MQ Series

IBM Websphere MQ Series requires capitalized queue names and does not allow slashes (QUEUEA). The name of the Queue Manager in MQ should match what the value of 'connection-factory' is or you will need to bind this name to JNDI. In our case we created a Queue Manager named "ConnectionFactory".

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

On your classpath you should have:

- **com.ibm.mq.pcf.jar**
- **mqcontext.jar**
- **com.ibm.mq.jar**
- **com.ibm.mqjms.jar**

Please note that the client jars differ between MQ 5.3 and MQ 6.0, but the 6.0 jars should be backward compatible. The jars are not open source, and are not provided by us. You will have to obtain them from your Websphere AS and MQ installations.

Also note that you may get the following exception when running MQ 6.0, which can be fixed by adding 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
```

1.2.4. Oracle Advanced Queuing (AQ)

Warning

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

For Oracle AQ you should set the parameters to:

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

and use the following properties:

```
<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"/>
```

You may notice the reference to the InitialContext factory. You only need this if you want to avoid OracleAQ registering its queues with an LDAP server. The **`AqinitialContextFactory`** references code in a plugin jar that you can find in the **`plugins/org.jboss.soa.esb.oracle.aq`** directory. The jar is called **`org.jboss.soa.esb.oracle.aq-4.2.jar`** and you will have to deploy it to the **`jbossesb.sar/lib`** directory.

When creating a Queue in Oracle AQ make sure to select a payload type of SYS AQ $_JMS_MESSAGE.

For a sample you can check the **`samples/quickstarts/helloworld_action/oracle-aq`** directory for an example **`jboss-esb.xml`** configuration file.

1.2.5. Tibco Enterprise Message Service (EMS)

When using Tibco EMS you should set the parameters to:

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

In your classpath you should have the client jars that ship with Tibco EMS, which are found in the tibco/ems/clients/java dir.

- jaxp.jar
- jndi.jar
- tibcrypt.jar
- tibjmsapps.jar
- tibrvjms.jar
- jms.jar

- jta-spec1_0_1.jar
- tibjmsadmin.jar
- tibjms.jar

TibcoEMS version 4.4.1 has tested with JBoss SOA

1.2.6. Extension Properties

By default the JNDI configuration used to retrieve the JMS resources will inherit all properties with names prefixed by "java.naming". Some JMS providers may specify properties that use a different naming prefix.

To support these properties we provide a mechanism to specify property prefixes for each provider, allowing properties using these additional prefixes to be inherited.

The prefixes are configured by defining the "jndi-prefixes" property on the associated jms-provider element, containing a comma separated list of the additional prefixes. The extension properties are also configured in the 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. Database Configuration

The SOA Platform uses a database for persisting Registry services, and the Message-Store.

Database scripts for each of these can be found under:

Service Registry: ESB_ROOT/install/juddi-registry/sql

Message-Store: ESB_ROOT/services/jbossesb/src/main/resources/message-store-sql

A few database types and their scripts are provided, and you should be able to easily create one for your particular database (if you do, please contribute it back to us).

For the Message-Store you will need to also update the data-source setting properties in the main config file jbossesb-properties.xml. The following are settings you will need to change, based on the connection information appropriate to your environment – these settings are found in the DBSTORE section of the file.

As long as there is a script for your database the SOA Platform will auto-create the schema's on startup. By default the SOA Platform is configured to use a JEE DataSource.

```
<properties name="dbstore">
  <property name="org.jboss.soa.esb.persistence.db.conn.manager"
    value="org.jboss.soa.esb.persistence.manager.J2eeConnectionManager"/>

  <!-- this property is only used if using the
       j2ee connection manager -->
```

```
    <property name="org.jboss.soa.esb.persistence.db.datasource.name"
      value="java:/JBossESBDS"/>
  </properties>
```

When running from the standalone bootstrapper use:

```
<properties name="dbstore">
 <!--  connection manager type -->
 <property name="org.jboss.soa.esb.persistence.db.conn.manager"
  value="org.jboss.soa.esb.persistence.manager.
StandaloneConnectionManager"/>
 <property name="org.jboss.soa.esb.persistence.db.conn.manager"
 <property name="org.jboss.soa.esb.persistence.db.connection.url"
  value="jdbc:hsqldb:hsql://localhost:9001/jbossesb"/>
 <property name="org.jboss.soa.esb.persistence.db.jdbc.driver"
  value="org.hsqldb.jdbcDriver"/>
 <property name="org.jboss.soa.esb.persistence.db.user" value="sa"/>
 <property name="org.jboss.soa.esb.persistence.db.pwd" value=""/>
 <property name="org.jboss.soa.esb.persistence.db.pool.initial.size"
  value="2"/>
 <property name="org.jboss.soa.esb.persistence.db.pool.min.size"
  value="2"/>
 <property name="org.jboss.soa.esb.persistence.db.pool.max.size"
  value="5"/>
 <property name="org.jboss.soa.esb.persistence.db.pool.test.table"
  value="pooltest"/>
 <property name="org.jboss.soa.esb.persistence.db.pool.timeout.millis"
  value="5000"/>
</properties>
```

Database Configuration Properties

`org.jboss.soa.esb.persistence.db.conn.manager`
: The database connection manager.

`org.jboss.soa.esb.persistence.db.datasource.name`
: The datasource name (used for JNDI lookup)

`org.jboss.soa.esb.persistence.db.connection.url`
: The database connection URL.

`org.jboss.soa.esb.persistence.db.jdbc.driver`
: JDBC Driver

`org.jboss.soa.esb.persistence.db.user`
: The database user

`org.jboss.soa.esb.persistence.db.pwd`
: The database password

`org.jboss.soa.esb.persistence.db.pool.initial.size`
: The initial size of database connection pool

`org.jboss.soa.esb.persistence.db.pool.min.size`
The minimum size of database connection pool

`org.jboss.soa.esb.persistence.db.pool.max.size`
The maximum size of database connection pool

`org.jboss.soa.esb.persistence.db.pool.test.table`
A table name (created dynamically by pool manager) to test for valid connections in the pool

`org.jboss.soa.esb.persistence.db.pool.timeout.millis`
The timeout period to wait for connection requests from pool

The Service Registry database information is contained in the juddi.properties file. You should consult the Service Registry section of this document for more detailed information on what settings and their values and how they effect the behavior of the SOA Platform.

JBoss server comes with a pre-installed hypersonic database (HSQLDB). The database can only be accessed in the same JVM. The data-source definition can be found in the jbossesb.sar/message-store-ds.xml.

Warning
Use of HSQLDB for production is not recommended.

1.4. Switching Databases

This section describes the steps required to change the datasource configuration from one database to another. The example demonstrates moving from the Hypersonic database to PostgreSQL.

The database Hypersonic (HSQLDB) is included in the default SOA Platform installation. This database has a number of limitations and is included only for testing and demonstration use. It is not suitable for production deployment and is not supported in that capacity.

These steps should be the same for any other database. Just replace the references to PostgreSQL files with those of the database you wish to use.

1.4.1. Step by Step

1. Remove deploy/hsqldb-ds.xml and add the following in a file named deploy/postgres-ds.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<datasources>
 <local-tx-datasource>
  <jndi-name>DefaultDS</jndi-name>
  <connection-url>jdbc:postgresql://host:port/database</connection-
url>
  <driver-class>org.postgresql.Driver</driver-class>
  <user-name>username</user-name>
  <password>password</password>
  <metadata>
   <type-mapping>PostgreSQL 7.2</type-mapping>
```

```
   </metadata>
   <check-valid-connection-sql>
    select count(*) from jbm_user
   </check-valid-connection-sql>
  </local-tx-datasource>
</datasources>
```

Modify the above to suite your needs, connection parameters and such. Make sure the name of the DS is the same though(DefaultDS)

2. Replace deploy/jbossesb.sar/juddi-ds.xml with the same configuration in the previous step (change the database name if needed). Ensure you keep the jndi-name(juddiDB)

3. Replace deploy/jbossesb.esb/message-store-ds.xml with the same configuration in step one (change the database name if needed). Ensure you keep the jndi-name(JBossESBDS).

4. Replace the database name in the 'message-store-sql' element in deploy/jbossesb.esb/jbossesb-service.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<server>
 <mbean code="org.jboss.internal.soa.esb.dependencies.DatabaseInitializer"
  name="jboss.esb:service=MessageStoreDatabaseInitializer">
  <attribute name="Datasource">java:/JBossESBDS</attribute>
  <attribute name="ExistsSql">select * from message</attribute>
  <attribute name="SqlFiles">
    message-store-sql/postgresql/create_database.sql
  </attribute>
  <depends>
    jboss.jca:service=DataSourceBinding,name=JBossESBDS
  </depends>
 </mbean>
</server>
```

5. Edit deploy/jbossesb.sar/esb.uddi.xml, and verify that it has a section that looks like this:

```
<entry key="juddi.isUseDataSource">true</entry>

<!-- jUDDI DataSource to use -->
<entry key="juddi.dataSource">java:/juddiDB</entry>

<!-- jUDDI database creation -->
<entry key="juddi.isCreateDatabase">true</entry>

<!--   <entry key="juddi.tablePrefix">JUDDI_</entry> -->
<entry key="juddi.databaseExistsSql">
    select * from ${prefix}BUSINESS_ENTITY
</entry>

<entry key="juddi.sqlFiles">
```

```
juddi-sql/postgresql/create_database.sql,juddi-sql/postgresql/
import.sql
</entry>
```

6. Replace deploy/jboss-messaging/hsqldb-persistence-service.xml with the postgres-persistence-service.xml from the version of JBM that you are running.

   This needs to match the same version and might not work if the versions mismatch. These files can be found in src/etc/server/default/deploy of a JBM distribution.

7. Copy the database driver to the servers lib directory and fire up the server.

1.5. Using a JSR-170 Message Store

The JBoss SOA Platform allows for multiple message store implementations via a plugin-based architecture. One of your alternatives to using the default database message store is to use a JSR-170 compliant Java Content Repository (JCR). The JCR implementation included is Apache Jackrabbit. To enable the JCR message store, add the following property to the "core" section of the **`jbossesb-properties.xml`** file in the root of the **`jboss-esb.sar`**:

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

This adds the JCR plugin to the list of available message stores. The JCR message store can use an existing repository via JNDI or can create a standalone instance locally on the application server. The following list of properties should be added in the "dbstore" section of jbossesb-properties.xml to configure repository access:

```
<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 - optional path in JNDI where the repository is found. If not specified, a new repository will be created based on the repository.xml located in the root of jbossesb.sar. In this case, repository data is stored in the **`JBossAS/server/{servername}/data/repository`** directory
- jcr.username - username for getting a repository session
- jcr.password - password for getting a repository session
- jcr.root.node.path - the path relative to the root of the repository where messages will be stored.

To quickly test that your JCR message store is configured properly, add the **`org.jboss.soa.esb.actions.persistence.StoreJCRMessage`** action onto an existing service. The action will attempt to store the current message to the JCR store.

1.6. Message tracing

It is possible to trace any and all Messages sent through the JBoss SOA Platform. This is often required for a number of reasons, including audit trail and debugging. Messages must be uniquely identified using the MessageID field of the Message header in order to be traced. This is referred to in the Programmers Guide. This is the only way in which Messages can be uniquely identified within the JBoss SOA Platform.

By default, JBoss SOA components (e.g., gateways, ServiceInvoker and load balancing) log all interactions with Messages using standard logger messages. The log messages will contain the header information associated with the Message, enabling correlation across multiple SOA Platform instances. These messages can be identified by looking for the following in your 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, you can enable a logging MetaData Filter, whose only role is to issue log messages whenever a Message is either input to an SOA Platform component, or output from it. This filter, org.jboss.internal.soa.esb.message.filter.TraceFilter, can be placed within the Filter section of the JBossESB configuration file, in conjunction with any other filters: it has no effect on the input or output Message. Whenever a Message passes through this filter, you will see the following log at info 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 ] )
```

TraceFilter will only log if the property org.jboss.soa.esb.messagetrace is set to **on/ON**. The default setting is **off/OFF**. If enabled it will log all Messages that pass through it. However you may enable finer grained control over which Messages are logged and which are ignored. To do this 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** will now be ignored by this filter.

1.7. Clustering and Fail-over support

JBoss SOA has support for fail-over of stateless services. You should consult the Programmers Guide for further details, but the pertinent issues to note are:

- Because ServiceInvoker hides much of the fail-over complexity from users, it only works with native SOA Platform Messages. Furthermore, in not all gateways have been modified to use the ServiceInvoker, so incoming SOA Platform-unaware messages to those gateway implementations may not always be able to take advantage of service fail-over. The included Release Notes contain more details regarding this.

- When the ServiceInvoker tries to deliver a message to your Service it may now get a choice of potentially multiple Endpoint References (EPR). In order to help it determine which one to select, you can configure a Policy. In the **`jbossesb-properties.xml`** you can set the org.jboss.soa.esb.loadbalancer.policy property. Right now three Policies are provided, or you can create your own.

  1. First Available: If a healthy ServiceBinding is found it will be used unless it dies, and it will move to the next EPR in the list. This Policy does not provide any load balancing between the two service instances.

  2. Round Robin: A typical Load Balance Policy where each EPR is hit in order of the list.

  3. Random Robin: This one is like the previous one, but the selection is randomized.

- The EPR list that the Policy uses may get smaller over time as dead EPRs will be 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 EPRs from the Registry. The org.jboss.soa.esb.registry.cache.life property defaults to 60000 milliseconds but can be set in the **`jbossesb-properties`** file.

- If none of the EPRs work then this would be a good situation for using the Message Redelivery Service.

- If you want to run the same service on more than one node in a cluster you have to wait for service registry cache re-validation before the service will be fully working in the clustered environment. You can setup this cache re-validation timeout in **`deploy/jbossesb.sar/jbossesb-properties.xml`**.

```
<properties name="core">
    <!-- 60 seconds is the default -->

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

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

1.8. Using OpenSSO with the SOA Platform

The JBoss SOA Platform includes the Open Web SSO project (OpenSSO) to simplify the implementation of a transparent single sign-on (SSO) service.

More information about OpenSSO, please visit its website at: *http://opensso.dev.java.net*

1.8.1. Installing and configuring OpenSSO in Tomcat

There is an known issue in deploying OpenSSO on the JBoss Enterprise SOA Platform but OpenSSO can be deployed to other web-containers for use with the SOA Platform.

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

Details concerning the deployment issue can be found at *https://jira.jboss.org/jira/browse/SOA-731*.

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

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

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

-

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

Example 1.1. Adding max size to JAVA_OPTS

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

Example 1.2. Adding max size to JAVA_OPTS

Procedure 1.1. Deploying OpenSSO to Tomcat

1. Download Tomcat from the Apache site: *http://tomcat.apache.org*
2. Unzip it to a directory. The following examples assume that this directory is **`/opt/tomcat`**
3. Edit **`/opt/tomcat/bin/catalina.sh`** (**`catalina.bat`** for Windows deployments) and add *`-Xmx1G`* to the JAVA_OPTS property. This specifies the maximum heap size of the JVM instance as one gigabyte.

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

   Example 1.3. Adding max size to JAVA_OPTS
4. Download **`opensso.zip`** (build 4.5) from OpenSSO website: *https://opensso.dev.java.net/public/use/index.html*
5. Unpack **`opensso.zip`** and copy **`opensso.war`** from **`deployable-war/`** to **`/opt/tomcat/webapps/`**
6. If you want to deploy JBoss Enterprise SOA Platform and Tomcat on the same machine you can update the Tomcat port in **`$tomcat/server.xml`** as below:

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

Example 1.4. Updating Tomcat port

7. Start Tomcat by running **`/opt/tomcat/bin/startup.sh`** (**`startup.bat`** for Windows deployments).
8. Open *http://localhost:8090/opensso* in a browser.
9. Click on **`Create Default Configuration`**.
10. Enter **`adminpass`** for the **`Default User[amAdmin]`** and **`ldappass`** for **`Default Agent [amldapuser]`**
11. Click on **`Create Configuration`**. This is cause OpenSSO to configure itself
12. Open *http://localhost:8090/opensso* again. Log in using the proper credentials. User Name is **`amAdmin`** and Password is the password you chose to go with amAdmin.

You can find further details of OpenSSO on Tomcat at this blog entry: *http://blogs.sun.com/JohnD/entry/how_to_install_tomcat_6*.

1.8.2. Configuring OpenSSO for the JBoss SOA Platform

The **`AuthContext`** class found in **`openssoclientsdk.jar`** performs the authentication. The following steps describe the configuration required to enable this integration.

Procedure 1.2. Configuring OpenSSO integration

1. Edit **`login-config.xml`**

   You need to edit the **`login-config.xml`** file located in the **`conf/`** directory of your server, eg. **`/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 1.5. Editing **`login-config.xml`** for OpenSSO

You need to have above configuration in the **login-config.xml** to provide the ability to integrate with OpenSSO. the 'orgName' and the 'moduleName' are the information that you configured in the OpenSSO system. The last property shows that where the **AMConfig.properties** file located.

2. Edit AMConfig.properties

   AMConfig.properties is located in the **conf/props/** directory of your server, eg. **/jbossas/server/default/conf/props/AMConfig.properties**

   By default, we configured it to **localhost**, **8080** port and the **opensso** context path. If you want to change it to your own configuration, or adopt an existed deployed OpenSSO, it is suggested that you use the **scripts/setup.sh** (**setup.bat** for Windows deployments) to do the configuration.

   The script is **/samples/fam-client/sdk/scripts/setup.sh** which is found in can be found in **opensso.zip**.Once you run it, you will simply have a screen as following:

```
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 **$opensso.zip/samples/fam-client/sdk/resources/AMConfig.properties**. For other information about opensso configuration, please conduct the opensso documentation at: *http://opensso.dev.java.net*.

After finishing above two steps, you are able to use the OpenSSOLogin module as a JAAS plugin provider.

You can use it as an identity provider to secure SOA Platform like this:

```
<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>
```

Registry

At the heart of all JBoss SOA Platform deployments is the registry. This is fully described in the JBoss SOA Platform Services Guide, where configuration information is also discussed. However, it is worth noting the following:

- When services run they typically place the EndPointReference (EPR) through which they can be contacted within the registry. If they are correctly developed, then services should remove EPRs from the registry when they terminate. However entries can be left within the registry by machine crashes or incorrectly developed services. These stale entries prevent the correct execution of subsequent deployments. In that case these entries may be removed manually. However, it is obviously important that you ensure the system is in a inactive state before doing so.
- If you set the optional remove-old-service tag name in the EPR to true then the ESB will remove any existing service entry from the Registry prior to adding this new instance. However, this should be used with care, because the entire service will be removed, including all EPRs.

Configuring Web Service Integration

The JBoss SOA Platform exposes Webservice Endpoints using the SOAPProcessor action. This action integrates the JBoss Webservices v2.x container into JBoss SOA, allowing you to invoke JBossWS Endpoints over any channel supported by JBoss SOA.

The SOAPProcessor action requires JBossWS 2.0.1.SP2 (native) to be properly installed on your JBoss SOA Server.

You should refer to the Programmers Guide for more details.

Default ReplyTo EPR

JBoss SOA uses Endpoint References (EPRs) to address messages to and from services. As described in the Programmers Guide, messages have headers that contain recipient addresses, sequence numbers (for message correlation) and optional addresses for replies, faults etc. Because the recommended interaction pattern for JBoss SOA is based on one-way message exchange, responses to messages are not necessarily automatic: it is application dependent as to whether or not a sender expects a response.

A reply address is an optional part of the header routing information which an application should set if necessary. When a response is required and the ReplyTo EPR has not been set, JBoss SOA supports default values for each type of transport. Some of these ReplyTo defaults require system administrators to configure JBoss SOA correctly.

- For JMS, it is assumed to be a queue with the same name as the one used to deliver the original request, prefixed with '_reply'.
- For JDBC, it is assumed to be a table in the same database with the same name as one used to deliver the original request, prefixed with '_reply_table'. The new table needs the same columns as the request table.
- For local and remote files no administration changes are required. Responses are written into the same directory as the request with a unique suffix to ensure that only the original sender will pick up the response.

ServiceBinding Manager

If you wish to run multiple JBoss SOA servers on the same machine, you may want to use the JBoss ServiceBinding Manager. The binding manager allows you to centralize port configuration for all of the instances you will be running. The JBoss SOA server ships with a sample bindings file in **`docs/examples/binding-manager/sample-bindings.xml`**.

The Jboss application server documentation contains detailed instructions on how to set up the ServiceBinding manager.

If you are using jboss-messaging as your JMS provider, your ServiceBinding manager configuration for jboss-messaging must match what is in remoting-service.xml.

Monitoring and Management

There are a number of options for monitoring and managing your ESB server. Shipping with the ESB are a number of useful JMX MBeans that help administrators monitor the performance of their server.

Under the jboss.esb domain, you should see the following MBean types:

- deployment=<ESB package name> – Deployments show the state of all of the ESB packages that have been deployed and give information about their XML configuration and their current state.
- listener-name=<Listener name> – All deployed listeners are displayed with information including their XML configuration, start time, maxThreads and state. The administrator has the option of initializing, starting, stopping and destroying a listener.
- category=MessageCounter – Message counters display all of the services deployed for a listener, each service's separate actions and counts of how many messages were processed, as well as the processing time of each message.
- service=<Service-name> - Displays statistics for each service including message counts, state, average size of message and processing time. The message counts may be reset and services may be stopped and started.

Additionally, JMS domain MBeans show statistics for message queues, which is useful for debugging or determining performance.

6.1. Monitoring and Management Console

The JBoss SOA Platform has its own monitoring and management console for SOA related properties (http://localhost:8080/jbossesb).

The JBoss SOA monitoring console gathers information on the performance of different services that are deployed and keeps historical state information over a period of time. The monitoring console allows users to get message counts by service, action, and node, as well as other information like processing time, number of failed messages, bytes transferred, and last successful and failed message date time.

The monitoring console is installed by default in the stand-alone JBoss SOA server. However, if you have need to install it manually then installing the JBoss ESB monitoring console is easy. The console uses HSQLDB as its database by default, so you can install with the steps of :

```
% cd tools/console/management-esb
% ant deploy
```

6.1.1. Alternative database usage

The console has also been tested with Oracle and MySQL as well as HSQLDB. It can be extended to use any JDBC/Hibernate-supported database.

In the management-esb directory there is a db.properties file. In order to change the database from HSQLDB to MySQL or Oracle, edit this file and change the db property to "mysql" or "oracle" respectively. You will also need to add your JDBC driver into the server/<instance>/lib directory of your application server. JBoss SOA ships with hsqldb.jar in this directory by default.

For MySQL, it also may be necessary to create the database "statistics" before deploying. Please look over the management-ds.xml for your database in the /management-esb/src/main/resources/<db> directory.

6.1.2. Collection Periods

The period of time between data collections is 10 minutes by default, but it can be set to any period of minutes that is desired. The default collection period can be changed at build time by changing the "pollMinuteFrequency" property in management-esb/db.properties, or by changing the PollMinuteFrequency property in the jboss.esb:service=DataFilerScheduler Mbean in the monitoring console or in jmx-console.

6.1.3. Console

The console requests and displays MBean information from each node within the ESB registry.

The console can be found at *http://localhost:8080/jbossesb/*.

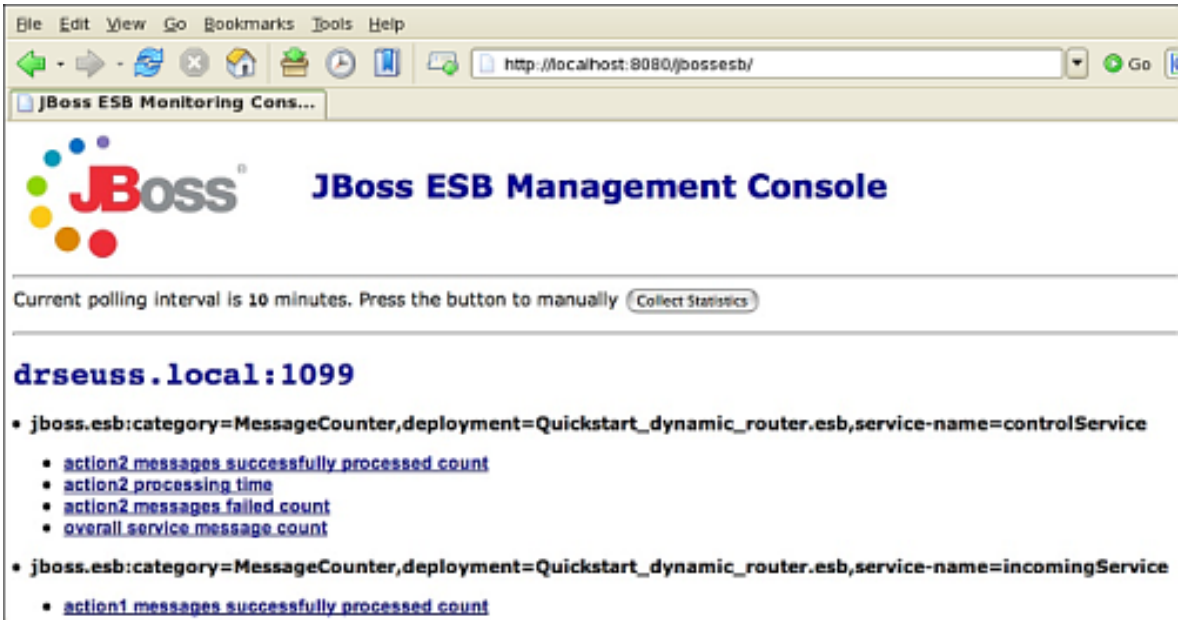

Figure 6.1. JBoss ESB Monitoring Console

6.1.4. Polling

The console's default polling period is 10 minutes. This can can be changed using the jmx-console. The **Collect Statistics** button at the top of the console page allows a user to force a statistics collection.

6.1.5. 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). If you select any of these options, you should see a screen that charts the count or time you have selected.

By default, the last 10 records are displayed. You can display more records by changing the display records text box or you can change the charting time period (graph over the last 5 minutes, hour, day, week, month, or graph all records).

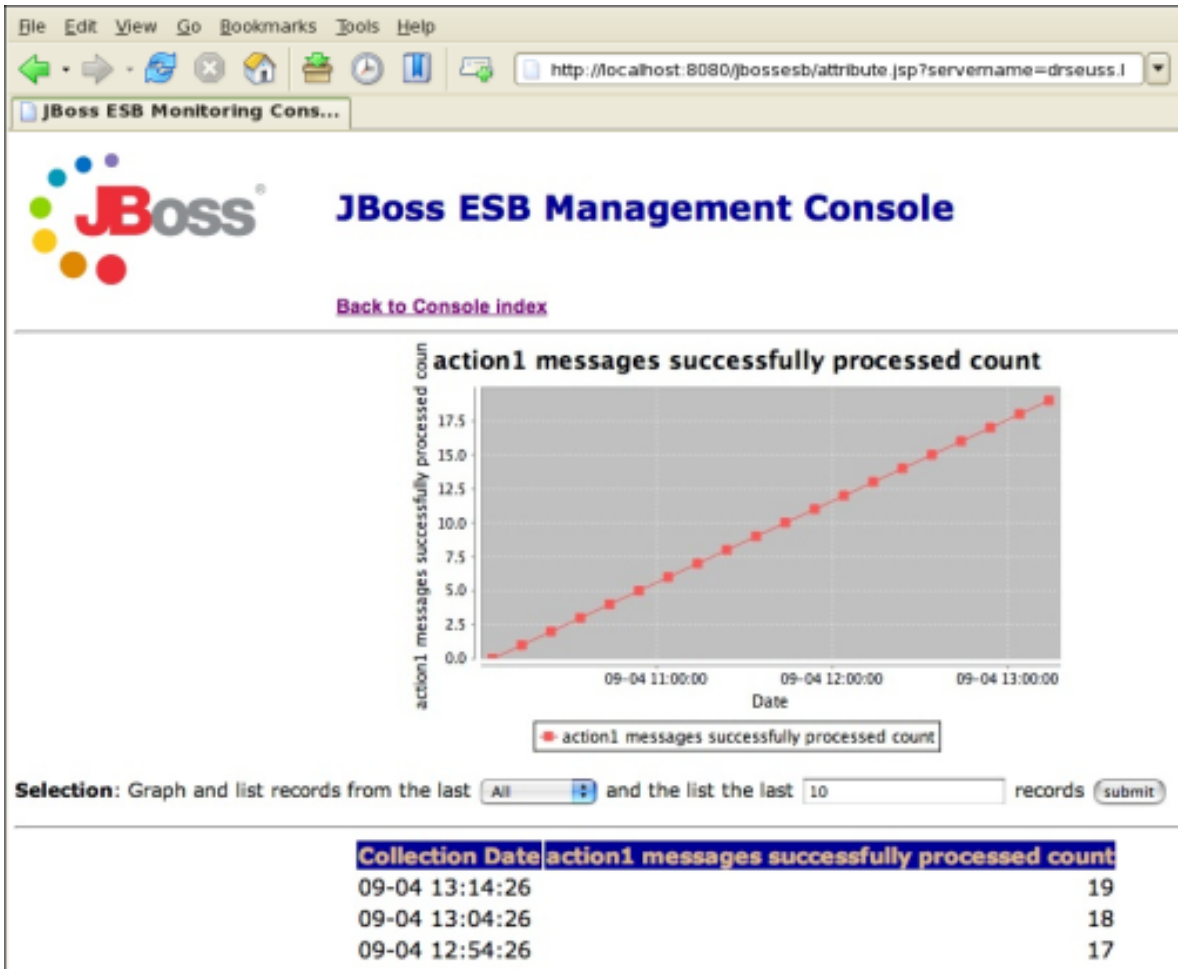

Figure 6.2. JBoss ESB Monitoring Console

6.1.6. Message Counter

The monitoring console also provides an overall counter which counts all messages that pass through the ESB. The MessageCounter keeps track of the successful and failed message counts, as well as time and date.

Figure 6.3. JBoss ESB Monitoring Console Message Counter

6.1.7. Transformations

For each Smooks Transformation that is registered, the monitoring console keeps track of the processed count for each transformation, processing time for each transformation, and the overall count for the transformation chain.

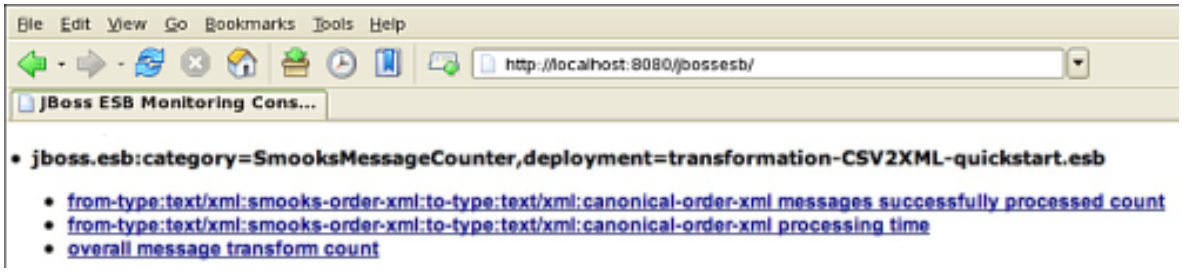

Figure 6.4. JBoss ESB Monitoring Console Transformations

6.1.8. Dead Letter Service

As has been mentioned in the Programmers Guide, the DeadLetterService (DLQ) can be used to store messages that cannot be delivered. This is a JBossESB service and can be monitored and inspected. Note, however, that the DLQ is not used if the underlying transport has native support, e.g., JMS. In which case you should inspect the JBossESB DLQ as well as any transport-specific equivalent.

6.2. Alerts

The JBoss Web Console (https://wiki.jboss.org/auth/wiki/WebConsole) is a utility within both the JBoss AS and the JBoss ESB Server that is capable of monitoring and sending alerts based off of JMX MBean properties. You can use this functionality to receive alerts for ESB-related events – such as the DeadLetterService counter reaching a certain threshold.

1. Configure ./deploy/mail-service.xml with your SMTP settings.
2. Change ./deploy/monitoring-service.xml – uncomment the EmailAlertListener section and add appropriate header related information.
3. Create a file ./deploy to serve as your monitor MBean.

```
<?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>
```

This MBean will serve as a monitor, and once the DeadLetterService counter reaches 5, it will send an e-mail to the address(es) specified in the monitoring-service.xml. Note that the alert is only sent once – once the threshold has been reached. If you want to be alerted again once resetting the counter, you can reset the alerted flag on your monitoring service MBean (in this case jboss.monitor:service=ESBDLQMonitor).

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

6.3. JON for SOA

An additional option that you have to monitoring and administering your JBoss SOA Platform server is the JBoss Operations Network product.

The JBoss Operations Network (JON) product provides inventorying, administration, monitoring, deployment and updating for JBoss-based middleware applications. This is performed using a centrally managed model with a customizable web-portal interface. Additional details on this product can be found at its website, *http://www.jboss.com/products/jbosson*.

“JON for SOA” is a standalone release of JON that includes functionality specially designed for the JBoss SOA Platform. This chapter provides an overview of that functionality and assumes a basic knowledge of the JBoss Operations Network product.

Important

Access to the JON console is not restricted to the local server like the embedded JBoss SOA Platform consoles are. This grants you greater freedom in its use, but also means you cannot rely on those restrictions to ensure the security of the JON console.

Adding your JBoss SOA Platform Server to the JON Inventory

Your JBoss SOA Platform server will appear in JON as a resource type of “JBossAS Server” with the description of “JBoss Enterprise SOA Platform”.

When you first try to access your SOA Platform server in JON you will see an error message because you have not yet provided the authentication details for a valid SOA Platform user.

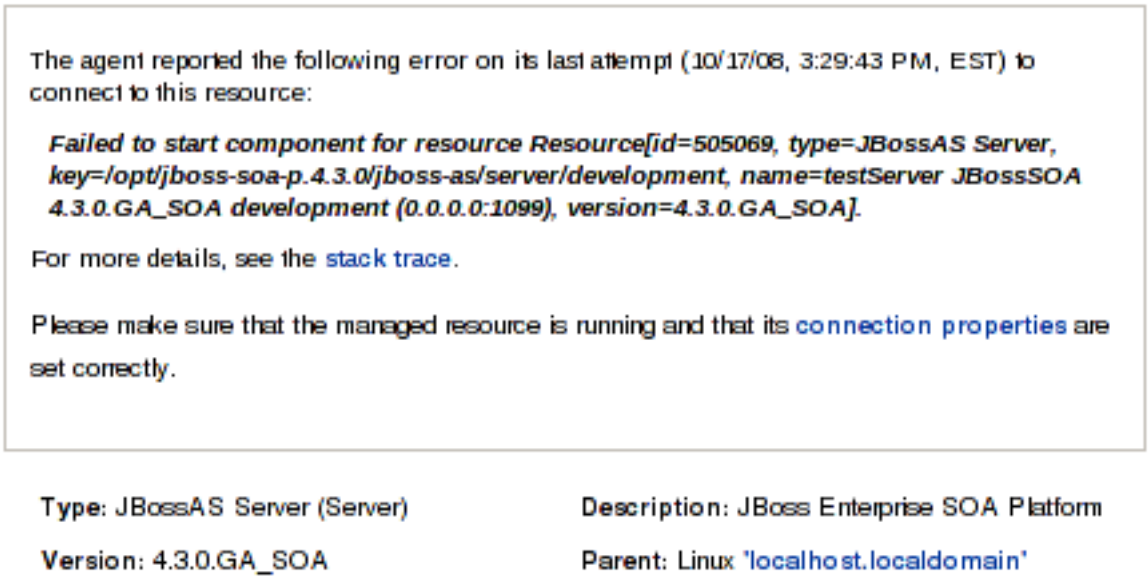

Figure 6.5. Error displayed when no or incorrect authentication information is supplied.

The user information is found in the **`conf/props/soa-users.properties`** file of the server profile in use. You enter this information as Principal and Credentials (username & password) in the server's Connection Properties. These details are accessed by selecting the server and then the **`INVENTORY`** tab. The error message also contains a shortcut link to the connection properties page.

JBoss SOA ESB Statistics

Once your JBoss SOA Platform server is correctly configured in JON you will find **JBoss ESB Statistics** listed under it in Resources on the **`MONITOR`** tab.

Clicking on **JBoss ESB Statistics** drills down into the ESB Statistics. At this level the figures displayed are an overall of the ESB instance.

If you click on the **JBoss ESB Deployment** item this take you to a list of all the deployed ESB packages on your server. No statistics are displayed at this level.

Selecting an ESB deployment will drill down into that deployment and display the statistics for it. From here you can also drill down into the details for the services and actions that make up the deployment.

The metrics collected and displayed varies depending on the ESB component. 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

All the standard JON functionality such as Alerts and Charts can be configured for an ESB Deployment, Service or Action for based on these statistics.

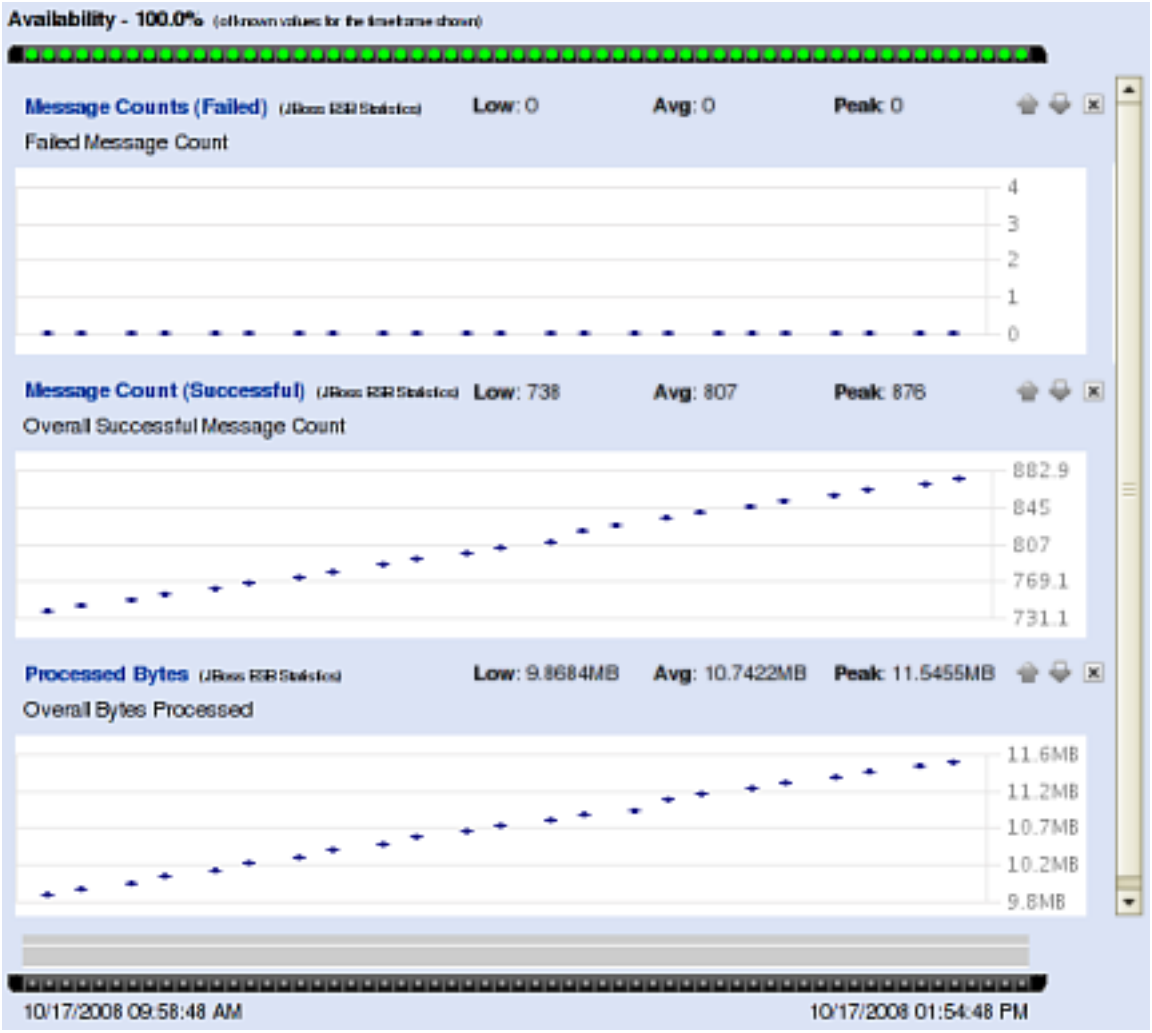

Figure 6.6. Displayed Metrics

Managing deployed ESB archives

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

A new ESB archive can be deployed by selecting **JBoss ESB Deployment** from the **Create New** menu. From the **Create New Resource** page you then specify the archive to deploy, where to deploy it to (normally your **`deploy`** directory) and whether to deploy as a zipped or exploded archive.

Existing archives can be deleted by checking them in the list of **Child Resources** and clicking **DELETE**.

The history of ESB deploy and delete requests is shown in this section as well.

Automatic Service Discovery

The JON Agent will automatically detect ESB archives which 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 from the inventory.

The default agent configuration only performs this service discovery once every 24 hours. You have two means to change this time period.

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

   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 6.7. Service discovery period setting in **`conf/agent-configuration.xml`**

2. Use JON console to edit the configuration

   JON Agents can be added to your inventory of server resources. Once added their configuration can be edited like any other inventoried resource. you can edit the **Service Discovery Period** value under the **CONFIGURE** tab of that resource. This does not require a restart of the Agent to take effect.

Unlike the SOA Platform embedded console there is no way to force the JON Console to perform an immediate collection of new data. Buttons such as the **Get Current Values** button in the **Metric Data** tab only update the display to reflect the most recently collected data. To get an immediate update you can set the collection period very low value like 30 seconds and then set the interval back afterwards. For performance reasons it is not recommened to lower the collection period significantly.

Hot Deployment

7.1. Server Mode

The JBoss SOA Platform supports "hot deployment". The server regularly checks the 'deploy' directory for new files to deploy. However it also checks already deployed files for specific changes. When these changes are detected the files are un-deployed by the server and the new files are deployed in their place. This is referred to as "hot re-deployment".

The specific changes that are monitored for vary by package type.

1. sar files

   The jbossesb.sar is hot deployable. It will redeploy when:

   - the timestamp of the archive changes, if the sar is a compressed archive.
   - the timestamp of the META-INF/jboss-service.xml changes, if the sar is in exploded form.

2. esb files

   Any *.esb archive will redeploy when

   - the timestamp of the archive changes, if the sar is a compressed archive.
   - the timestamp of the META-INF/jboss-esb.xml changes, if the esb is in exploded form.

   Our actions have life-cycle support. Upon hot re-deployment it will go down gracefully, finishing active requests and not accepting any more incoming messages until it is back up. All of this occurs automatically. If you want to update just one action, you can use Groovy scripting to modify an action at run-time (see the groovy QuickStart at *http://wiki.jboss.org/wiki/Wiki.jsp?page=JBossESBQuickStart*).

3. rule files

   There are two options to refresh rule files (drl or dsl)

   - redeploy the jbrules.esb archive
   - turn on the 'ruleReload' in the action configuration (see JBossESBContentBasedRouting at *http://wiki.jboss.org/wiki/Wiki.jsp?page=JBossESBContentBasedRouting*). Now if a rule file *changes* it will be reloaded.

4. transformation files

   There are two options to refresh transformation files

   - redeploy the esb archive in which the transformation file resides.
   - send out a notification message over JMS(topic) using the esb-console. The Smooks processors will receive this event and reload.

5. Business Process Definitions

New versions of jBPM Business Process Definitions can be deployed to the jbpm database using the jBPM Eclipse plugin. The new version will be used by new process instances. Existing process will finish their life-cycle on the previous definition. For more details please refer to the jBPM documentation.

7.2. Standalone (bootstrap) mode

The bootstrapper does not deploy esb archives. You can only have one jboss-esb.xml configuration file per node. It will monitor the timestamp on this file and it will reread the configuration if a change occurs. To updates rules you will have to use the 'ruleReload', to update transformation you need to send out a Smooks JMS notification using the esb-console. And finally to update BPDs you can follow the same process mentioned above.

Contract Publishing

Integrating to certain JBoss SOA endpoints may require information about that endpoint and the operations it supports. This is particularly the case for Webservice endpoints exposed via the SOAPProcessor action (see JBoss SOA Message Action Guide).

8.1. "Contract" Application

For this purpose, we bundle the "Contract" application with JBoss SOA [1]. This application is installed by default with the ESB (after running "ant deploy" from the install directory. [2].

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

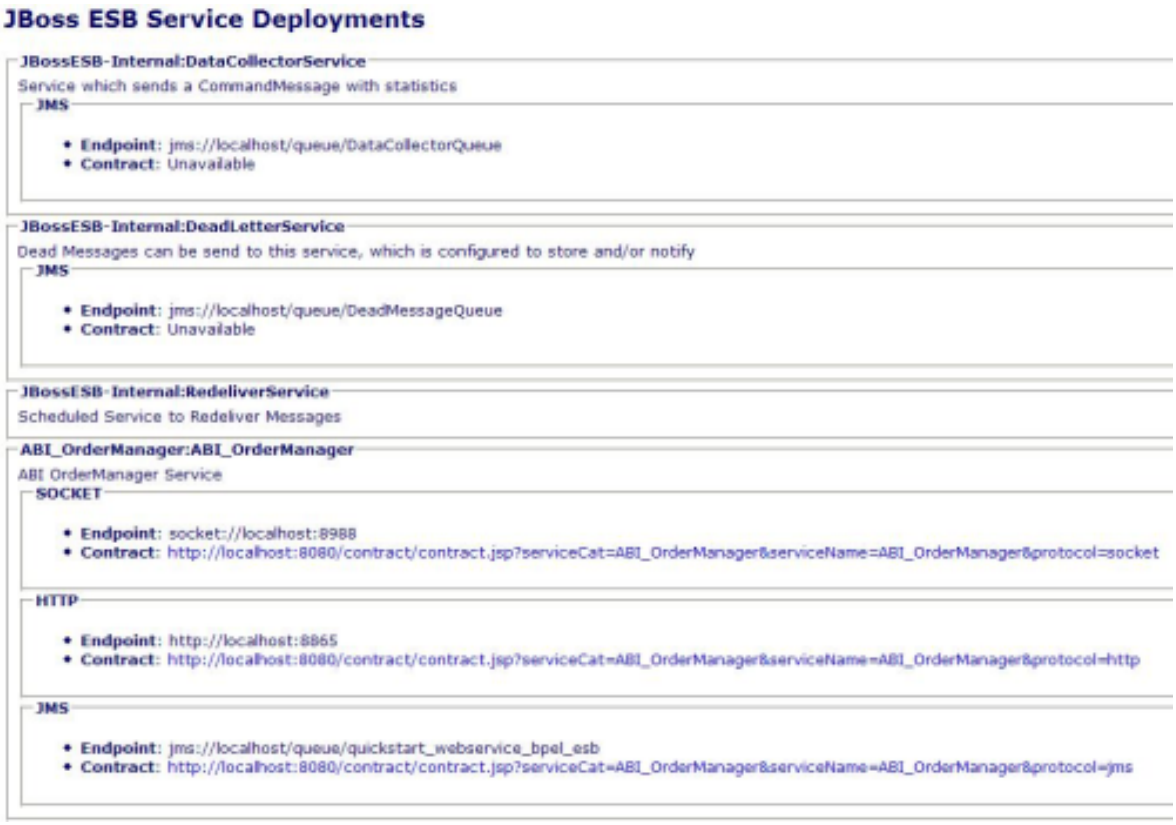

Figure 8.1. JBoss ESB "Contract" Application

As you can see, it groups the endpoint according to Service with which they are associated (servicing). Another thing you'll notice is how some of them have an active "Contract" hyperlink. The ones visible here are for Webservice endpoints exposed via the SOAPProcessor. This hyperlink links off to the WSDL.

8.2. Publishing a Contract from an Action

JBossESB discovers endpoint contracts based on the action pipeline that's configured on a Service. It looks for the first action in the pipeline that publishes contract information. If none of the actions

[1] This application is only being offered as a Technical Preview. 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 undeploy the default Contract application. Just remove contract.war from the default/deploy folder of your JBoss SOA Server.

publish contract information, then the Contract application just displays "Unavailable" on Contract for that endpoint.

An Action publishes contract information by being annotated with the org.jboss.internal.soa.esb.publish.Publish annotation as follows (using the SOAPProcessor as an example):

```
<xslthl:annotation>@Publish(WebserviceContractPublisher.class)</
xslthl:annotation>

public class SOAPProcessor extends AbstractActionPipelineProcessor {
    //TODO: implement
}
```

SOAPProcessor code as an example: *http://anonsvn.labs.jboss.com/labs/jbossesb/trunk/product/services/soap/src/main/java/org/jboss/soa/esb/actions/soap/SOAPProcessor.java*

You then need to implement a "ContractPublisher" (org.jboss.soa.esb.actions.soap.ContractPublisher), which just requires implementation of a single method:

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

Example WebserviceContractPublisher code as an example: *http://anonsvn.labs.jboss.com/labs/jbossesb/trunk/product/services/soap/src/main/java/org/jboss/soa/esb/actions/soap/WebserviceContractPublisher.java*

jBPM_Console

9.1. Overview

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

Appendix A. Revision History

Revision History
Revision 1.0 Fri Sep 5 2008 DarrinMison*dmison@redhat.com*
Initial Creation