Administration Guide

For JBoss ESB administrators

by JBoss ESB Development Team with Community Contributions

Configuration

1.1. Applying Configuration Changes

Many of the administrative sections refer to modifications which need to be applied to the `jbossesb-properties.xml` configuration file, specifically those which have a global impact on the system. The location of this configuration file depends on how the system is being run.

Within a server environment this file will usually be located at `server/PROFILE/deployers/esb.deployers/jbossesb-properties.xml`, while standalone clients will access the configuration file directly through their classpath.

1.2. Stand-Alone Server

If you wish to run the JBossESB server on the same machine as JBossAS, then you should look at the *Configuring Multiple JBoss Instances On One Machine* [http://www.jboss.org/community/wiki/ConfiguringMultipleJBossInstancesOnOnemachine] Wiki page.

1.3. JBossESB JMS Providers

The JBossESB supports a number of JMS providers. Currently we have successfully tested JBoss Messaging, JBossMQ, ActiveMQ and Websphere MQ Series (version 5.3 and 6.0). We recommend JBoss Messaging. At this time we know of no reasons why other JMS implementations should not also work, but have not been able to verify this.

Warning

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.

To configure `JMS Listeners` and `JMS Gateways` to listen to *queues* and *topics*, specify the following parameters in the `jbossesb-listener.xml` and `jbossesb-gateway.xml` configuration files:

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

- destination-name

Important

Read *Section 1.3.8, "JMS and Java Connector Architecture (JCA)"* before configuring your JMS Provider.

Important

Be sure to include the chosen JMS provider's client JAR files in the class-path.

Important

In the following sections we will assume that your JMS provider runs on 'localhost', that the connection-factory is 'ConnectionFactory', that we are listenening to a destination-type 'queue' and that it's name is 'queue/A'.

Note

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

When using JMS, JBossESB utilizes a connection pool to improve performance. By default the size of this pool is set to 20, but can be over-ridden by setting the org.jboss.soa.esb.jms.connectionPool property in the transports section of the JBossESB configuration file. Likewise, if a session cannot be obtained initially, JBossESB will keep retrying for up to 30 seconds before giving up. This time can be configured using the org.jboss.soa.esb.jms.sessionSleep property.

1.3.1. Max Sessions Per Connection

The JBoss ESB's JmsConnectionPool pools JMS Sessions and is used by all JMS based components – JMS Listeners, JMS Couriers, JMS Router etc.

Some JMS providers limit the number of JMS Sessions per connection. This means that the JMS Components in JBoss ESB need to support a control mechanism for the maximum number of sessions created from each JMS Connection managed by a single JmsConnectionPool instance. This is done simply be specifying one or both of the following properties in the JNDI configuration of the JMS Component (JMS Provider/Bus, JMSRouter etc):

- max-sessions-per-connection: This is the maximum total number of Sessions allowed per connection i.e. XA + non-XA Session instances. This value defaults to the maximum number of

JMS Sessions allowed for a JmsConnectionPool as a whole, which defaults to 20 (as configured in the jbossesb-properties.xml file).

- max-xa-sessions-per-connection: This is the maximum number of XA Sessions allowed per connection i.e. XA only. This value defaults to the value of max-sessions-per-connection.

So if neither of the above parameters are configured, the JmsConnectionPool will create a single JMS Connection and create all JMS Sessions off that Connection instance.

These configurations should be made as generic property configurations on the JMS Provider configuration e.g.

```
<jms-provider ...>

   <property name="max-sessions-per-connection" value="5" />
   <property name="max-xa-sessions-per-connection" value="1" />

   <!-- And add providers.... -->

</jms-provider>
```

1.3.2. JBossMQ and JBossMessaging

Important

Read *Section 1.3.8, "JMS and Java Connector Architecture (JCA)"* before configuring your JMS Provider.

The settings for JBossMQ and JBossMessaging are identical and 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/myqueue"
```

Note

For JBossMQ you should have jbossmq-client.jar in your classpath. Not that this jar is included in jbossall-client.jar, which can be found in lib/ext. For JBossMessaging

it should be jboss-messaging-client.jar While -for now- the JBossMQ is the default JMS provider in JBossAS, you can also use JBoss Messaging. Instructions for installing JBoss Messaging can be found on the project website, *http://docs.jboss.org/jbossmessaging/docs/userguide-1.4.0.GA/html/installation.html*.

1.3.2.1. JBoss Messaging Clustering configuration

Configuring JBoss Messaging in a clustered setup gives you loadbalancing and failover for JMS. Since this capability has changed between different versions of JBoss Messaging and may continue to do so, you should consult the relevant JBoss Messaging documentation (*http://docs.jboss.org/jbossmessaging/docs/userguide-1.4.0.SP1/html_single/index.html#conf.connectionfactory.attributes.loadbalancingfactory*).

Note

This functionality is continually changing as **JBoss Messaging** evolves. To learn how to configure it, consult the relevant **JBoss Messaging** documentation.)

1.3.3. ActiveMQ

Important

Read *Section 1.3.8, "JMS and Java Connector Architecture (JCA)"* before configuring your JMS Provider.

For 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. We tested with version 4.1.0-incubator.

1.3.4. Websphere MQ Series

Important

Read *Section 1.3.8, "JMS and Java Connector Architecture (JCA)"* before configuring your JMS Provider.

From a JBoss ESB perspective, configuring Websphere MQ providers is simple enough. You have 2 options:

1. As a JCA Provider using the Websphere MQ JCA Adapter through the <jms-jca-provider> configuration.
2. As a Standard JMS Provider through the <jms-provider> configuration.

1.3.4.1. Websphere MQ JMS Provider Configurations

The following is an example of a <jms-jca-provider> configuration:

```
<jms-jca-provider                    name="WMQ-JCA"                    connection-
factory="MyAppXAConnectionFactory" adapter="wmq.jmsra.rar"
      jndi-context-factory="com.ibm.mq.jms.context.WMQInitialContextFactory">

   <property name="max-xa-sessions-per-connection" value="1" />

   <jms-bus busid="quickstartGwChannel">
     <jms-message-filter dest-type="QUEUE" dest-name="QUEUE1_JMS" transacted="true"/>
   </jms-bus>

   <activation-config>
      <property name="queueManager" value="TQM" />
      <property name="channel" value="Q1CONN" />
      <property name="hostName" value="localhost" />
      <property name="port" value="1414" />
      <property name="transportType" value="CLIENT" />
   </activation-config>

</jms-jca-provider>
```

The above configuration is an example of how to configure a gateway provider through the <jms-jca-provider> configuration. The main things to note of specific relevance to Websphere MQ are:

1. The adapter name "wmq.jmsra.rar". This rar file must be present in the server's deploy folder.

2. The initial context factory "com.ibm.mq.jms.context.WMQInitialContextFactory".

3. The "queueManager" configuration in the <activation-config>. You need to configure a Queue Manager in your Websphere MQ installation. All your Queues (all destinations) are configured under your Queue Manager.

4. The "channel" configuration in the <activation-config>. A Queue Manager can be configured with a number of different types of "channels". Clients connect through "Server Connection" channels. A default server connection channel is added as part of the Websphere MQ installation ("SYSTEM.DEF.SVRCONN"), but it is a good idea to specify your own dedicated channels, especially if you are using extended (XA) transations. More on this later.

5. The "transportType" configuration in the <activation-config>. Websphere MQ supports transport types "Client" and "Binding".

We said that the above <jms-jca-provider> configuration is an example of how to configure a gateway provider. That's not strictly true; it's just a provider configuration. It is the <jms-listener> configuration that determines the gateway/message-aware characteristics of a listener. What is true to say about the above configuration however is that, as far as Websphere MQ is concerned, it is only of use as a provider to a gateway listener i.e. would not work as a Message-Aware listener for Websphere MQ . This is because it does not specify an appropriate JNDI Provider URL (that can be used by the ServiceInvoker) for routing ESB messages to the destinations (buses) defined on the provider. Note the <activation-config> configures the JCA adapter for getting messages from the destinations. It does not specify any information used for delivering messages to the destinations. An example of a configuration that could be used as a provider for a Message-Aware listener would be:

```
<jms-jca-provider                    name="WMQ-JCA"                    connection-
factory="MyAppXAConnectionFactory" adapter="wmq.jmsra.rar"
      jndi-URL="localhost:1414/CHANX"
      jndi-context-factory="com.ibm.mq.jms.context.WMQInitialContextFactory">

   <property name="max-xa-sessions-per-connection" value="1" />

   <jms-bus busid="quickstartEsbChannel">
     <jms-message-filter dest-type="QUEUE" dest-name="QUEUE2_JMS" transacted="true"/>
   </jms-bus>

   <activation-config>
      <property name="queueManager" value="TQM" />
      <property name="channel" value="Q2CONN" />
      <property name="hostName" value="localhost" />
      <property name="port" value="1414" />
      <property name="transportType" value="CLIENT" />
   </activation-config>
```

```
</jms-jca-provider>
```

Creating a standard JMS provider configuration is more or less the same:

```
<jms-provider name="JMS" connection-factory="MyAppConnectionFactory"
          jndi-URL="localhost:1414/CHAN1"
          jndi-context-factory="com.ibm.mq.jms.context.WMQInitialContextFactory">

   <jms-bus busid="quickstartGwChannel2">
      <jms-message-filter dest-type="QUEUE" dest-name="QUEUE3_JMS" />
   </jms-bus>

</jms-provider>
```

What you should note with the standard provider is that there's no adapter and <activation-config> configurations. Listeners getting messages from the destinations defined in a standard provider do not use a JCA Adapter Inflow to receive the messages. Instead, they need to use JNDI to lookup the destination and get the messages. This effectively means that for Websphere MQ, the jndi-URL must always be specified (remember that for JCA it is only required for destinations that service a Message-Aware listener).

1.3.4.2. JMSRouter Configuration for Websphere MQ

The following is an example of a JMSRouter configuration for routing messages to Websphere MQ from inside an ESB action pipeline:

```
<action name="routeToORDERSQueue" class="org.jboss.soa.esb.actions.routing.JMSRouter">
                                                <property name="jndi-context-
factory" value="com.ibm.mq.jms.context.WMQInitialContextFactory"/>
   <property name="jndi-URL" value="wmqserver:1414/CHANX"/>
   <property name="connection-factory" value="WMQConnectionFactory"/>
   <property name="jndiName" value="ORDERS"/>
</action>
```

1.3.4.3. Websphere MQ and XA Transactions

If your Websphere MQ endpoint is to participate in JTA/XA Transactions you will need to install the "Extended Transactional Client" bundle from IBM. This bundle may not be part of the default set of client jars provided with your Websphere MQ installation. If not, contact your IBM representative to get these jar files. Once you have the Extended Client bundle, install the .jar files on your ESB/Application Server's classpath, as well as on the classpath of any external client applications.

Before you can use extended transactions in WMQ, you need to *configure XA Connection Factories in your WMQ JNDI namespace* [http://publib.boulder.ibm.com/infocenter/wasinfo/v6r0/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/umj_pjcfm.html]. Once you have configured the XA Connection Factories, you can reference them using their JNDI name in the "connection-factory" property of the JNDI connection property.

The following is an example showing how the *<jms-jca-provider>* configuration would look for an Extended Transaction Client configuration, where the XA Connection Factory configured in the WMQ JNDI namespace is called "WMQXAConnectionFactory". Inflow related configurations (adapter, <activation-config> etc) were intentionally omitted as they are not relevant to the Extended Client configuration:

```
<jms-jca-provider name="WMQ" connection-factory="WMQXAConnectionFactory"
                  jndi-URL="wmqserver:1414/CHANXA_SEND"
                  jndi-context-factory="com.ibm.mq.jms.context.WMQInitialContextFactory">

   <property name="max-xa-sessions-per-connection" value="1" />

   <jms-bus busid="ordersGwChannel">
      <jms-message-filter dest-type="QUEUE" dest-name="ORDERS" transacted="true"/>
   </jms-bus>

   <activation-config>
      <!--
      Used by inflow... not relevant to client
      See section on JMS and JCA.
      -->
   </activation-config>

</jms-jca-provider>
```

The following is an example of using an the same XA Connection Factory on the JMSRouter:

```
<action name="routeToORDERSQueue" class="org.jboss.soa.esb.actions.routing.JMSRouter">
                                                    <property                name="jndi-context-
factory" value="com.ibm.mq.jms.context.WMQInitialContextFactory"/>
   <property name="jndi-URL" value="wmqserver:1414/CHANXA_SEND"/>
   <property name="connection-factory" value="WMQXAConnectionFactory"/>
   <property name="jndiName" value="ORDERS"/>
   <property name="max-xa-sessions-per-connection" value="1"/>
   <!-- etc... -->
</action>
```

Important

An important point to note about Websphere MQ and XA is that it does not support both getting and putting messages concurrently on the same Queue Manager Channel. For this reason, it is a good idea to configure a dedicated channel for each component that gets or puts messages into a Websphere MQ destination in the context of XA transactions.

1.3.4.4. Enabling Tracing for Websphere MQ JCA Adapter

The WebSphere MQ resource adapter lets you configure diagnostic trace as a property on the resource adapter, or by setting JVM system properties. With the JBoss ESB/Application, the recommended method is to use JVM system properties. On systems that use run.sh, you can set these by editing the JBoss run.conf file (in the /bin directory) and appending the following onto the end of the file:

```
# Settings to enable WebSphere MQ resource adapter trace
JAVA_OPTS="$JAVA_OPTS -DtraceEnabled=true -DtraceDestination=wmq_jca.trc
   -DtraceLevel=10 -DlogWriterEnabled=false"
```

Client logging can also be enabled by setting the additional MQJMS_TRACE_LEVEL property:

```
# Settings to enable WebSphere MQ resource adapter and client trace
JAVA_OPTS="$JAVA_OPTS -DtraceEnabled=true -DtraceDestination=wmq_jca.trc
   -DtraceLevel=10 -DlogWriterEnabled=false -DMQJMS_TRACE_LEVEL=base"
```

1.3.4.5. Enabling Tracing for Websphere MQ Java Client

This is done programmatically by calling the "enableTrace" static method on the com.ibm.mq.MQEnvironment class.

1.3.4.6. Websphere MQ Gotchas and Checklist

1. If using `XA connections`, set the max-xa-sessions-per-connection property to `1`. (For more information on how to configure this property, see the "Maximum Sessions per Connection" section of this chapter.)

2. If using `XA connections`, be sure to configure a dedicated Queue Manager Channel for each component that gets or sets messages into a Websphere MQ destination. See earlier section titled "Websphere MQ and XA Transactions".

3. Make sure the GET and PUT properties on all your Websphere MQ destinations is not set to Inhibit. This can be set by default when the destination is created and results in the ESB not being able to put/get messages.

4. Check that the following items are present on the class-path:

   - `com.ibm.mq.pcf.jar`
   - `mqcontext.jar`
   - `dhbcore.jar`
   - `com.ibm.mq.jar` (client `JAR`)
   - `com.ibm.mqjms.jar` (client `JAR`)
   - See the *section below* if participating in JTA/XA Transactions.

   If using the Websphere MQ v7.0 Client jar files you will also need to add the following jar files to your classpath:

   - `com.ibm.mq.commonservices.jar`
   - `com.ibm.mq.headers.jar`
   - `com.ibm.mq.jmqi.jar`

Note

The client jars differ between MQ 5.3 and MQ 6.0. However the 6.0 jars should be backward compatible. The jars are not open source, and are therefor not provided by us. You will have to obtain them from your WAS and MQ installs.

Warning

The following exception may be encountered when running **Websphere MQ 6.0**:

```
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
```

This is a permissions issue. To fix this problem, add the user responsible for running the **JBoss Enterprise Service Bus** to the `mqm group`.

If using JCA Inflow, you will need to configure the "adapter" attribute on the <jms-jca-provider> configuration and ensure that the JCA Adapter RAR file is in the ESB/Application Server's deploy folder:

```
<jms-jca-provider name="WMQ" ... adapter="wmq.jmsra.rar">

   <jms-bus busid="ordersGwChannel">
      <jms-message-filter dest-type="QUEUE" dest-name="ORDERS" transacted="true"/>
   </jms-bus>

   <activation-config>
      <!-- etc... -->
   </activation-config>

</jms-jca-provider>
```

Also, be sure to read the *section on JMS and JCA*, as well as how to configure JCA <connection-factories> and AdminObjects for your JMS Destinations on JBoss Application Server.

1.3.5. Oracle AQ

Important

Read *Section 1.3.8, "JMS and Java Connector Architecture (JCA)"* before configuring your JMS Provider.

1. For Oracle AQ you should set the parameters to:

```
connection-factory="QueueConnectionFactory"
```

2. 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"/>
```

3. Optionally specify a database connection url:

```
<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.jdbc.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 above example can be used to connect to Oracle Real Application Cluster (RAC).

Note

You may notice the reference to the InitialContext factory. You only need this is if you want to avoid OracelAQ to register its queues with an LDAP. 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.

Note

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

Note

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

1.3.6. Red Hat MRG

Important

Read *Section 1.3.8, "JMS and Java Connector Architecture (JCA)"* before configuring your JMS Provider.

1. For Red Hat MRG you should set the parameters to:

```
<property name="jndi-prefixes" value="connectionfactory,.destination"/>
<property name="jndi-connection-factory" value="org.apache.qpid.jndi.PropertiesFileInitialContextFactory"/>
<property name="connectionFactory.qpidConnectionFactory" value="amqp://guest:guest@clientid/virtualHost?brokerlist='tcp://localhost:5672'"/>
<property name="destination.[queueName]Queue" value="direct://amq.direct//[queueName]?routingkey=[routingkeyname]"/>
```

2. In your classpath in your classpath you should have the Apache Qpid qpid-common and qpid-client JARs.

   - `qpid-common-0.6.jar`
   - `qpid-client-0.6.jar`

Note

We tested with version 0.6.

1.3.7. Tibco EMS

Important

Read *Section 1.3.8, "JMS and Java Connector Architecture (JCA)"* before configuring your JMS Provider.

1. For 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="myqueue"
```

2. 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`

Note

We tested with version 4.4.1.

1.3.8. JMS and Java Connector Architecture (JCA)

JBoss ESB supports a JMS JCA transport, with a dedicated JMS JCA Provider configuration (<jms-jca-provider>). This transport allows JCA message Inflow in 2 ways:

1. JMS Gateway using a JCA connection to the JMS Provider.
2. ESB Aware JMS Message Listener using a JCA connection to the JMS Provider.

The following is a very simple example of a <jms-jca-provider> configuration:

```
<jms-jca-provider name="JBossMessaging" connection-factory="XAConnectionFactory">

   <jms-bus busid="ordersGwChannel">
      <jms-message-filter dest-type="QUEUE" dest-name="queue/orders" transacted="true"/>
   </jms-bus>

   <activation-config>
```

```
        <property name="dLQMaxResent" value="5"/>
    </activation-config>

</jms-jca-provider>
```

Note

The <jms-jca-provider> element configures how the JMS JCA Inflows (Gateway or ESB Aware Listener) receive messages, *but also* configures how JMS clients deliver messages to them. While these configurations are made in the same location and may appear to be one and the same, they can often be different and lead to confusion for some users. Therefore, it is important for users to be aware of which <jms-jca-provider> configurations relate to Inflow (server side) Vs client connection configuration (client side). Read below for more details.

The <jms-jca-provider> configuration configures 2 separate (but related) things, either explicitly or through the application of defaults:

1. The JCA Inflow configuration properties (server side). These configrations configure the JCA Gateway and JCA Listeners listed above. The configurations include:
   - The JCA Adapter Name . This is configured as an attribute on the <jms-jca-provider> element.
   - The JCA Provider Adapter JNDI. This is configured as an attribute on the <jms-jca-provider> element.
   - The JCA Endpoint Class. This is configured as an attribute on the <jms-jca-provider> element.
   - Transacted flag. This is configured as an attribute on the <jms-jca-provider> element.
   - Message Type. This is configured as an attribute on the <jms-jca-provider> element.
   - JCA Bridge. This is configured as an attribute on the <jms-jca-provider> element.
   - The JCA Adapter activation configuration. This is configured in the <activation-config> element inside the <jms-jca-provider>.

   Note that the JCA Activation configuration also extracts some configuration properties (for the JCA Inflow) from the <jms-bus> and <jms-message-filter> that are nested inside the <jms-jca-provider> element. These include the destination type, destination name and message selector.
2. The JMS client connection details (client side) used to deliver messages to the JMS JCA Inflows (Gateway or ESB Aware Listener). This Information is used to generate the EPR for clients connecting to the JMS JCA Inflow. The configurations include:

- The JMS Connection factory to be used by the client. This is configured as an attribute on the <jms-jca-provider> element.
- The JMS JNDI properties to be used by the client for connecting to the JMS Provider. These are configured as an attributes on the <jms-jca-provider> element ("jndi-*").
- The JMS bus endpoint destination configurations. These are configured on the <jms-message-filter> elements inside the <jms-bus> within the <jms-jca-provider> element.

Note

Users should be aware of which <jms-jca-provider> configurations relate to message Inflow configuration (server side) and which relate to the client message delivery configuration (client side). Read above for more details.

It is important to ensure that the JMS client connection configurations (outlined above) are *not configured to connect through JCA managed resources*. The JMS client connection configurations should connect directly to the JMS Provider and not through JCA managed resources. This is important because this information is built into the EPRs used by the ESBs `ServiceInvoker` class, as well as other components that use the ESB's inbuilt JMS Connection Pooling. It is important to avoid a situation where the Inbuilt Connection Pooling is pooling JCA managed connections.

Note

Do not configure the JMS client configurations on <jms-jca-provider> to connect through JCA managed resources. Read above for more details.

The current list of components that use the ESBs inbuilt JMS Connection Pooling are:

1. Routing Actions that use the `ServiceInvoker` class e.g. the Static and Content Based Routers etc.
2. Most/All Gateway Listeners delivering messages to an Action Pipeline that is serviced by an ESB Aware JMS Listener that uses a JCA JMS Provider. Most of these Gateway Listeners deliver messages using the `ServiceInvoker` class.
3. The JMS Router.

These components should all connect directly to the JMS Provider i.e. not through local JCA managed resources.

1.3.9. Extension properties

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

Note

In order to support these properties we provide a mechanism through which the property prefixes can be specified 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>
```

1.4. FTP Configuration

Most configuration options are set on the FTP EPR and described in the Programmers Guide. However, the following are set at the global scope in the jbossesb-properties file:

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

  When transmitting files via FTP, JBossESB sends them over with one file name which prevents them being processed, before renaming them in order that they can be processed. Unfortunately some FTP servers retain locks on the file during the time it is written and then renamed, preventing the rename from happening. If this happens, JBossESB will attempt to rename the file the defined number of times (default 10), sleeping in between each attempt, before finally generating an error message if the file cannot be renamed.

1.5. Database Configuration

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

Database scripts for each of these can be found under:

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

Service Registry: The service registry is now jUDDI v 3.1.0, which does not use SQL scripts but uses the persistence layer to initialize the database schema.

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 ESB 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 script for your database the ESB will auto-create the schema's on startup. By default JBossESB 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"/>

   <!-- FIXME: is this a typo ? -->
   <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>
```

Properties

org.jboss.soa.esb.persistence.db.conn.manager
: the db 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
: this is the db connection url for your database.

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

org.jboss.soa.esb.persistence.db.user
: db user

org.jboss.soa.esb.persistence.db.pwd
: db password

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

org.jboss.soa.esb.persistence.db.pool.min.size
: minimum size of db connection pool

org.jboss.soa.esb.persistence.db.pool.max.size
: maximum size of db 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
: timeout period to wait for connection requests from pool

The Service Registry database information is contained in the `esb.juddi.xml` 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 ESB.

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.

1.5.1. Switching Databases

This section describes the steps to move from using the default hypersonic database to postgres. These steps should be the same for any other database. Just replace postgres with the database you want to switch to.

Procedure 1.1. Changing the database to PostgreSQL

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-registry.sar/juddi-ds.xml` with the same configuration in the previous step (change the database name if needed).
   Again make sure the 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).

Again make sure the 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 the `persistence.xml`, located in either `jbossesb-registry.sar/META-INF/` or `jbossesb-registry.sar/juddi_config/META-INF/`. hibernate.dialect property must be set to the type of database that is going to be used as the data source. (It may, for example, be `org.hibernate.dialect.PostgreSQLDialect`.)

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 it 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.

Procedure 1.2. Changing the database to DB2

- In addition to the standard procedure to change the database (listed above), you may need additional steps if you are changing to DB2 9.7 or above. You also need to configure the table space size - at least 8k page size table space is needed. Here are some sample settings:

```
create bufferpool soabp immediate size 1000 pagesize 8k;
create tablespace soats pagesize 8k managed by AUTOMATIC STORAGE BUFFERPOOL soabp;
create temporary tablespace soatempts pagesize 8k managed by AUTOMATIC STORAGE BUFFERPOOL soa
```

1.6. Using a JSR-170 Message Store

JBossESB allows for multiple message store implementations via a plugin-based architecture. As an alternative to the default database message store, a JSR-170 (Java content repository) message store may be used. The JCR implementation included with JBossESB is Apache Jackrabbit. To enable the JCR message store, add the following property to the "core" section of jbossesb-properties.xml 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.JCRMessageStorePlugin"/>
```

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"/>
```

Properties

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 gettging a repository session

jcr.root.node.path
: The path relative to the root of the repository where messages will be stored.

An easy test for whether the JCR message store is configured properly is to 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.

1.7. Message Tracing

It is possible to trace any and all Messages sent through JBossESB. This may be important for a number of reasons, including audit trail and debugging. In order to trace Messages you should ensure that they are uniquely identified using the MessageID field of the Message header: as mentioned in the Programmers Guide, this is the only way in which Messages can be uniquely identified within the ESB.

By default, JBossESB components (e.g., gateways, ServiceInvoker and load balancing) log all interactions with Messages through standard logger messages. Such log messages will contain the entire header information associated with the Message which will enable correlation across multiple JBossESB instances. You can identify these messages 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 ESB 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: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 ] )

TraceFilter.onInput ( 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 ] )
```

TraceFilter will only log if the property org.jboss.soa.esb.messagetrace is set to on/ON (the default setting is off/OFF). By default, if enabled it will log all Messages that pass through it. However, for finer grained control 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 (the default is off/OFF). Once enabled, those Messages with a Property of org.jboss.soa.esb.message.unloggable set to yes/YES will be ignored by this filter.

1.8. Clustering and Fail-Over Support

Beginning with JBossESB 4.2, there is now 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 necessarily only works with native ESB Messages. Furthermore, not all gateways have been modified to use the ServiceInvoker, so incoming ESB-unaware messages to those gateway implementations may not always be able to take advantage of service fail-over.
- When the ServiceInvoker tries to deliver a message to our Service it may get a choice of potentially multiple EPRs now. 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'. 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. Typical Load Balance Policy where each EPR is hit in order of the list.
  3. Random Robin. Like the other Robin but then random.
- The EPR list the Policy works with may get smaller over time as dead EPRs will be removed from the (cached) list. 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' can be set in the jbossesb-properties file, and is defaulted to 60,000 milliseconds. What if none of the EPRs work at the moment? This is where we may use Message Redelivery Service.
- If you would like to run the same service on more than one node in a cluster, wait until the service registry cache re-validates. Configure the cache re-validation time-out in the `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 you set the org.jboss.soa.esb.failure.detect.removeDeadEPR property to true, then whenever 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: for example, if the service represented by the EPR is simply overloaded and slow to respond then it may be excluded from future users. Therefore, if you allow ServiceInvoker to remove EPRs it is possible orphan services (ones that eventually receive no further interactions) may result and you may have to restart them.

Registry

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

- When services run they typically place the 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, machine crashes, or incorrectly developed services, may leave stale entries within the registry that 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 quiescent 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

JBoss ESB 4.10 exposes Webservice Endpoints for through the SOAPProcessor action. This action integrates the JBoss Webservices v2.x container into JBossESB, allowing you to invoke JBossWS Endpoints over any channel supported by JBossESB. See the Programmers' Guide for more details.

Important

The SOAPProcessor action requires JBossWS 2.0.1.SP2 (native) or higher to to be properly installed on your JBoss Application Server (v4.2.x.GA).

Default ReplyTo EPR

JBossESB uses Endpoint References (EPRs) to address messages to/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 within JBossESB 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.

As such, a reply address (EPR) is an optional part of the header routing information and applications should be setting this value if necessary. However, in the case where a response is required and the reply EPR (ReplyTo EPR) has not been set, JBossESB supports default values for each type of transport. Some of these ReplyTo defaults require system administrators to configure JBossESB correctly.

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

ServiceBinding Manager

If you wish to run multiple ESB servers on the same machine, you may want to use JBoss ServiceBinding Manager. The binding manager allows you to centralize port configuration for all of the instances you will be running. The ESB server ships with a sample bindings file in docs/examples/binding-manager/sample-bindings.xml. Chapter Ten of the JBoss appplication server documentation contains instructions on how to set up the ServiceBinding manager.

`remoting-service.xml` – If you are using jboss-messaging as your JMS provider, please note that what you specify in your ServiceBinding manager xml for jboss-messaging configuration 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 on their XML configuration, the start time, maxThreads, state, etc. The administrator has the option of initialising/starting/stopping/destroying a listener.

category=MessageCounter
: Message counters break all of the services deployed for a listener down into their separate actions and give counts of how many messages were processed, as well as the processing time of each message.

service-name=<Service name>
: Displays statistics per-service (message counts, state, average size of message, processing time, etc). The message counts may be reset and services may be stopped and started.

Note

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

6.1. Monitoring and Management

JBossESB provides management and monitoring through Embedded JOPR: (*http://localhost:8080/admin-console*).

The JBossESB monitoring console gathers information on the performance of different ESB services that are deployed . As of JBoss ESB 4.2.0.GA, 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. As of JBoss ESB 4.6, the previous ESB monitoring tool has been deprecated.

The monitoring console is installed automatically in the stand-alone ESB server and JBossAS.

Below is a screenshot of the console. The console requests MBean information from each node within the ESB registry, and then displays it back.

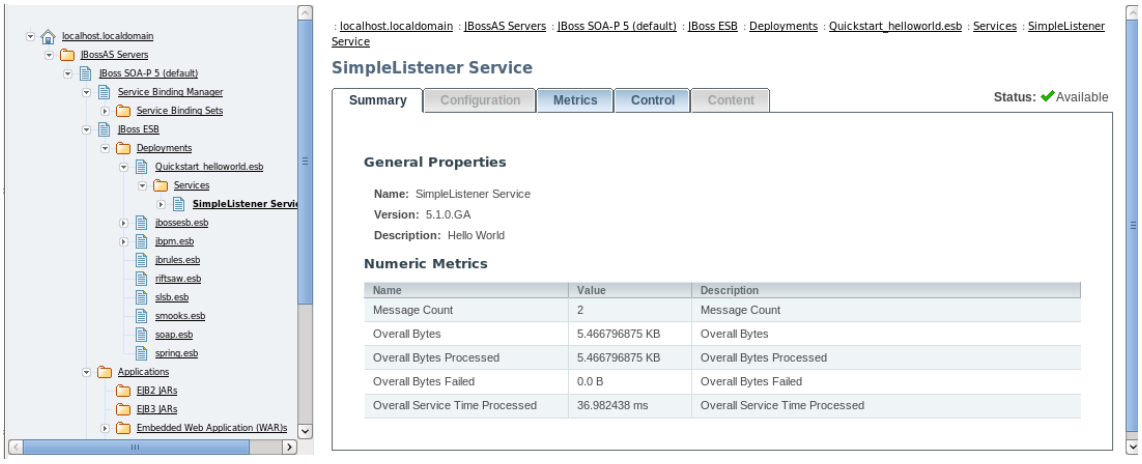

Figure 6.1. JBoss Monitoring and Management Console

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).

See below:

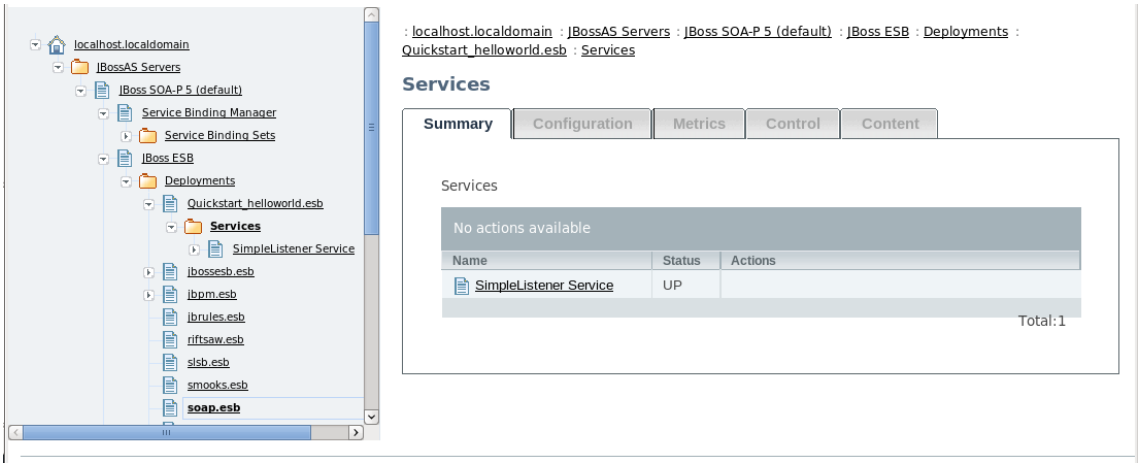

Figure 6.2. Services

6.1.2. The Message Counter

Seen above, 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.

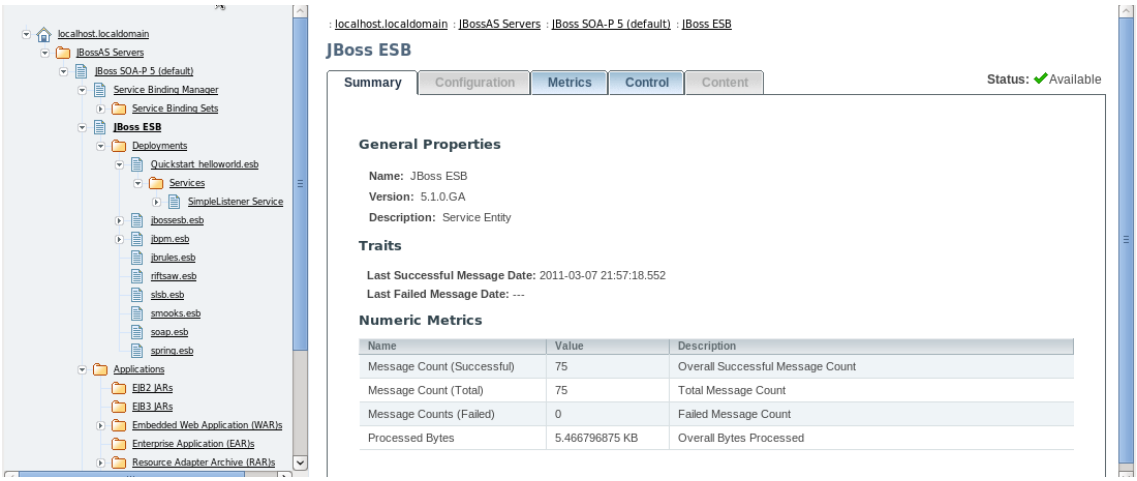

Figure 6.3. Monitoring and Management Console Message Counter

6.1.3. Transformations

For each Smooks Transformation that is registered, a Mbean keeps track of the processed count for each transformation, processing time for each transformation, and the overall count for the transformation chain. You can see this information in the jmx-console.

6.1.4. DeadLetterService

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.

6.1.5. Alerts

The JBoss Web Console (*http://community.jboss.org/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).

Note

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

Hot Deployment

JBossAS as well as the JBossESB-Server are always checking the 'deploy' directory for new files to deploy. So we're really talking about hot redeployment. So here is what you have to do to make it redeploy an existing deployment for the different components.

1. SAR files

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

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

2. ESB files

   Any *.esb archive will redeploy when

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

   Our actions have lifecycle support, so upon hot deployment it goes down gracefully, finishes active requests, and does not accept any more incoming messages until it is back up. All of this can be done by simply redeploying the .esb archive. If you want to update just one action, you can use groovy scripting to modify an action at runtime (see the groovy QuickStart: *http://wiki.jboss.org/wiki/Wiki.jsp?page=JBossESBQuickStart*.)

3. rule files

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

   a. redeploy the jbrules.esb (see 2)

   b. turn on the 'ruleReload' in the action config (see *http://wiki.jboss.org/wiki/Wiki.jsp?page=JBossESBContentBasedRouting*). Now if a rule file *changes* it will be reloaded. .) After doing so, if a rule file is changed, it will be re-loaded.

4. transformation files

   The only way to refresh transformation files is to redeploy the esb archive in which the transformation file resides.

5. Business Process Definitions

   When using jBPM new Business Process Definitions can be deployed. From within the jBPM eclipse plugin you can deploy a new definition to the jbpm database. New process instances will get the new version, in flight processes will finish their life cycle on the previous definitions. For details please see the documentation on jBPM.

7.1. 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'. And finally to update BPDs you can follow the same process mentioned above.

Contract Publishing

8.1. Overview

Integrating to certain ESB 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 Message Action Guide).

8.2. "Contract" Application

Important

This application is only being offered as a Technical Preview. It will be superseded in a later release.

For this purpose, we bundle the "Contract" application with the ESB. This application is installed by default with the ESB (after running "ant deploy" from the install directory of the distro). Note that the Contract application is also bundled inside the JBossESB 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 ESB/App Server.

It can be accessed through the URL *http://localhost:8080/contract/*.

Figure 8.1, “The Contract Application” is a screenshot of this application.

Figure 8.1. The 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.

8.3. 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 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):

```
@Publish(WebserviceContractPublisher.class)
public class SOAPProcessor extends AbstractActionPipelineProcessor {
    ...
}
```

See the SOAPProcessor code as an example at *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);
```

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

jBPM

9.1. jBPM Console

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.

9.2. jBPM Message and Scheduler service

The jBPM integration within ESB now support additional Message and Scheduler services, distinct from those offered natively by jBPM. In addition to the standard jBPM configurations we now also support a JMS based Message Service, driven using JCA inflow, and three additional Scheduling Services, based on JBoss Messaging, hornetq and quartz.

The configuration, as shipped by default within ESB, uses the jBPM JobExecutor and the database implementations of the Message and Scheduler service.

```
<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>
```

In order to utilize the alternative services it is necessary to replace the active configurations with the versions specific to your requirements.

The configuration for the alternative services can be found within the jbpm.esb/config directory

1. jbpm.esb/config/hornetqscheduler for configuring the hornetq based message and scheduler services
2. jbpm.esb/config/jmsscheduler for configuring the JBoss Messaging based message and scheduler services
3. jbpm.esb/config/quartzscheduler for configuring the JMS message service and a quartz based scheduler service.

The configuration files within the appropriate directory should be used to replace the active configurations within the jbpm.esb directory, remembering to remove the .config suffix from each. It should also be noted that only one of the quartz message queue service definitions should be used, either hornetq-jms.xml, jbm-queue-service.xml or jbmq-queue-service.xml, depending on which JMS implementation is currently in use.

Performance Tuning

To learn about performance tuning, go to *http://community.jboss.org/wiki/JBossESBPerformanceTuning*.

Appendix A. Revision History

Revision History

Revision 1	Fri Jul 16 2010	DavidLe Sage<dlesage@redhat.com>, DarrinMison<dmison@redhat.com>

Initial conversion from OpenOffice ODT files.