JBoss ESB 4.5 GA

Administration Guide

JBESB-AG-2/11/09

Legal Notices

The information contained in this documentation is subject to change without notice.

JBoss Inc. makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. JBoss Inc. shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Java™ and J2EE is a U.S. trademark of Sun Microsystems, Inc. Microsoft® and Windows NT® are registered trademarks of Microsoft Corporation. Oracle® is a registered U.S. trademark and Oracle9™, Oracle9 Server™ Oracle9 Enterprise Edition™ are trademarks of Oracle Corporation. Unix is used here as a generic term covering all versions of the UNIX® operating system. UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company Limited.

Copyright

Software Version

JBoss ESB 4.5 GA

Restricted Rights Legend

Contents

Table of Contents

About This Guide

What This Guide Contains

The Administration Guide contains important information on how to configure and manage installations of JBoss ESB 4.5 GA.

Audience

This guide is most relevant to system administrators who are responsible for managing and deploying JBoss ESB 4.5 GA installations.

Prerequisites

None.

Organization

This guide contains the following chapters:

- **Chapter 1, Configuration:** How to configure JBossESB and the services it supports.
- **Chapter 2, Registry:** How to configure the Registry.
- **Chapter 3, Configuring the Web Services Integration:** How to configure Web Services within JBossESB.
- **Chapter 4, Default ReplyTo EPR:** A description of how default ReplyTo EPRs are selected.
- **Chapter 5, ServiceBinding Manager:** How to deploy multiple JBossESB instances on the same machine.
- **Chapter 6, Monitoring and Management:** An overview of the monitoring and management capabilities within JBossESB.
- **Chapter 7, Hot Deployment:** Describes the hot deployment capabilities.
- **Chapter 8, Contract Publishing:** An overview of the contract publishing capabilities.

Documentation Conventions

The following conventions are used in this guide:

Convention	Description
Italic	In paragraph text, italic identifies the titles of documents that are being referenced. When used in conjunction with the Code text described below, italics identify a variable that should be replaced by the user with an actual value.
Bold	Emphasizes items of particular importance.
Code	Text that represents programming code.
Function | **Function**	A path to a function or dialog box within an interface. For example, "Select File | Open." indicates that you should select the Open function from the File menu.
( ) and |	Parentheses enclose optional items in command syntax. The vertical bar separates syntax items in a list of choices. For example, any of the following three items can be entered in this syntax: ***persistPolicy (Never | OnTimer | OnUpdate | NoMoreOftenThan)***
Note: **Caution**:	A note highlights important supplemental information. A caution highlights procedures or information that is necessary to avoid damage to equipment, damage to software, loss of data, or

Table 1 Formatting Conventions

Additional Documentation

In addition to this guide, the following documents are available in the JBoss ESB 4.5 GA documentation set:

1. **JBoss ESB 4.5 GA** *Trailblazer Guide*: Provides guidance for using the trailblazer example.
2. **JBoss ESB 4.5 GA** *Getting Started Guide*: Provides a quick start reference to configuring and using the ESB.
3. **JBoss ESB 4.5 GA** *Programmers Guide*: How to use JBossESB.
4. **JBoss ESB 4.5 GA** *Release Notes*: Information on the differences between this release and previous releases.
5. **JBoss ESB 4.5 GA** *Services Guides*: Various documents related to the services available with the ESB.

Contacting Us

Questions or comments about JBoss ESB 4.5 GA should be directed to our support team.

Chapter 1

Configuration

Standalone 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** Wiki page.

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.

> Note: 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?

JMSListeners and JMSGateways can be configured to listen to a Queue or Topic. For this you can use 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` and `destination-name.` Furthermore you will need to add the client jms jars of the JMS-provider you want to use to the classpath.

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.

JBossMQ or JBossMessaging

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/A"
```

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.

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.

ActiveMQ

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.

Websphere MQ Series

For Websphere MQ Series you should set the parameters to:

```
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"
```

Note: Websphere likes all CAPS queue names and no slashes (QUEUEA), and the name of the Queue Manager in MQ should match what the value of 'connection-factory' is (or bind this name to JNDI). In our case we created a Queue Manager named “ConnectionFactory”.

On your classpath you should have

com.ibm.mq.pcf.jar

mqcontext.jar

and the client jars:

com.ibm.mq.jar

com.ibm.mqjms.jar

Please note that 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.

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 jbossesb to the mqm group:

Note that for 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

Explanation: There is a problem with user permissions or access.

Tip: Make sure the user accessing MQ Queue Manager is part of the mqm group.

Oracle AQ

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"/>
```

or 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.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)))) "/>
```

The above example can be used to connect to Oracle Real Application Cluster (RAC).

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 that 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.

Tibco EMS

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="<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

We tested with version 4.4.1.

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.

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"/>
  ...
</jms-provider>
```

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.

Database Configuration

The ESB uses 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 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.StandaloneConnectionMan
ager"/>

  <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>
```

Property	Setting
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.

Note: Use of HSQLDB for production is not recommended.

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.

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).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 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 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.

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:

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

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 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.

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.

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 you have to wait for service registry cache revalidation before the service is fully working in the clustered environment. You can setup this cache revalidation timeout in deploy/jbossesb.sar/jbossesb-properties.xml:

```
<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.

Chapter 2

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.

Chapter 3

Configuring Web Service Integration

JBoss ESB 4.5 GA 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.

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).

Chapter 4

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.

Chapter 5

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. Two notes :

- *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*.

Chapter 6

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=<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.

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

Monitoring and Management Console

JBossESB has its own monitoring and management console for ESB related properties (http://localhost:8080/jbossesb).

The JBossESB monitoring console gathers information on the performance of different ESB services that are deployed and keeps historical state information over a period of time. 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.

The monitoring console is installed automatically in the stand-alone ESB server and JBossAS. However, if you have need to install it manually then installing the JBoss ESB monitoring console is fairly easy. The console uses hsqldb as a database by defalt, so you can install with the steps of :

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

Point your browser to http://localhost:8080/jbossesb

Alternative database usage

If you'd like to use a database other than hsqldb as a back-end, the console has also been tested with Oracle and MySQL – and could 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 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.

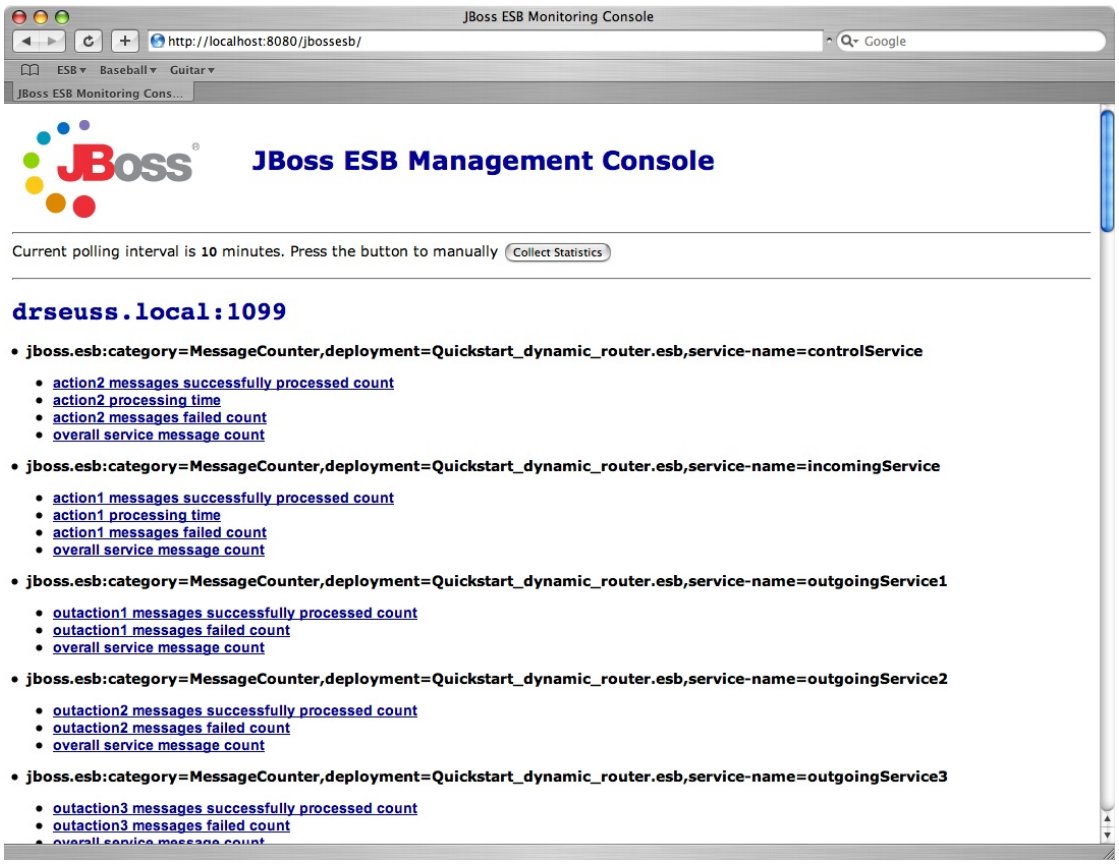
Collection Periods

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.

Console

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

Below is a screenshot of the console. The console requests MBean information from each node within the ESB registry, and then displays it back. Processing times are shown with a chart and a time sorted list, which all other data is displayed with a time sorted list.



Polling

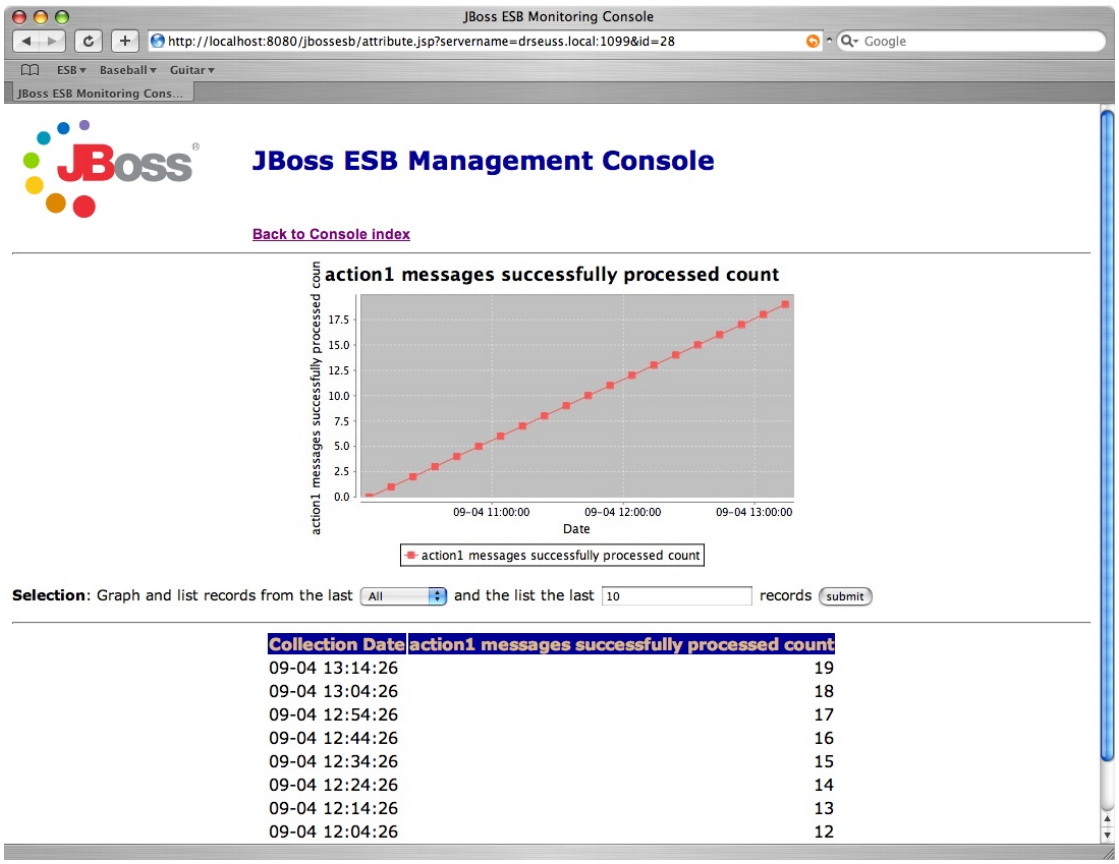
The console's polling default is 10 minutes, which can be changed at build time or through the jmx-console. The “Collect Statistics” button shown in the header allows a user to force a statistics collection.

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.

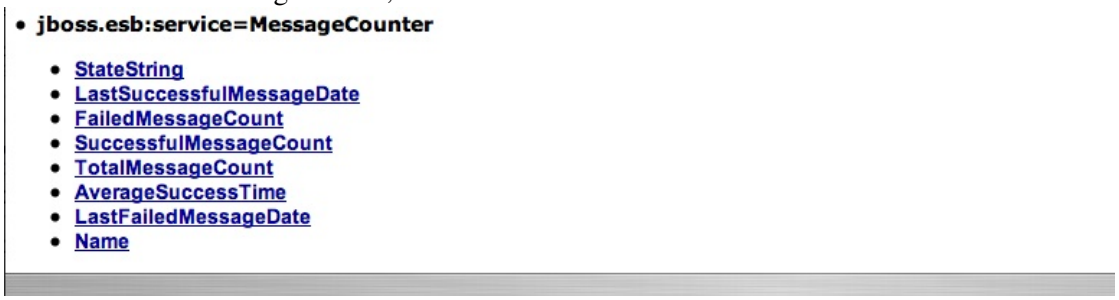
By default, the last 10 records are displayed. You can display more records by changing the display records text box or you can change the charting time period (graph over the last 5 minutes, hour, day, week, month, or graph all records).

See below:



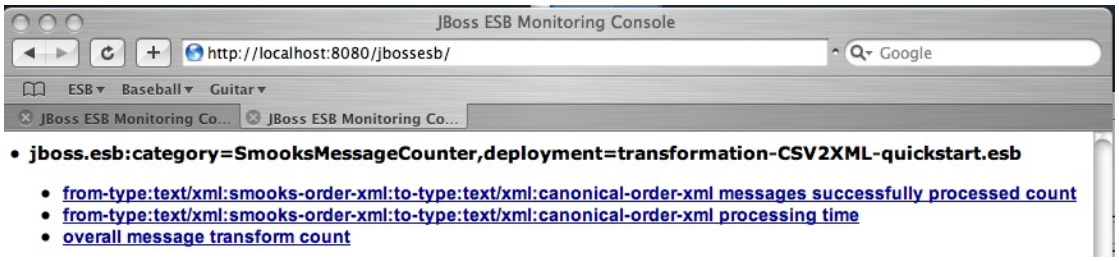
MessageCounter

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.



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.



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.

Alerts

> The JBoss Web Console 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.

File: ./deploy/DeadLetterQueue_Monitor-service.xml

```
<?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 see the Wiki pages.

Chapter 7

Hot Deployment

Server mode

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).

3. rule files

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

- redeploy the jbrules.esb (see 2)
- turn on the 'ruleReload' in the action config (see JBossESBContentBasedRouting). Now if a rule file *changes* it will be reloaded.

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.

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.

Chapter 8

Contract Publishing

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).

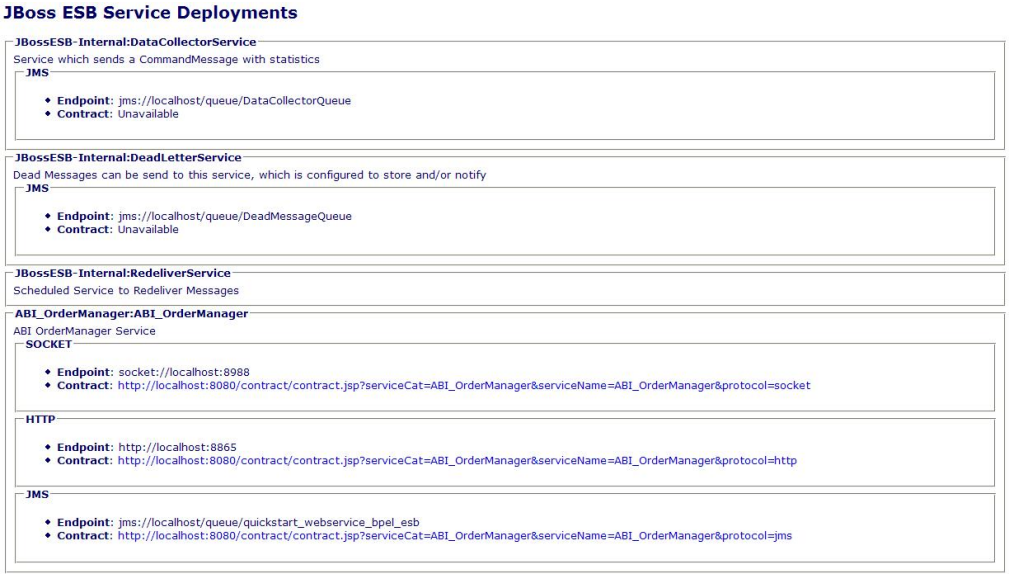
"Contract" Application

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

It can be accessed through the following URL:

http://localhost:8080/contract/

The following is a screenshot of this application.



[1]**NOTE:** 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 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.

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.

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.

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.

Chapter 9

jBPM Console

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.

Chapter 10

Performance Tuning

Overview

In this chapter we shall look at ways of optimizing the performance of JBossESB and tuning it for your specific environment. Before doing this, however, you should realize that as with any system, there is always a trade-off to be made between performance and reliability. The out-of-the-box configuration for JBossESB is tuned for maximum reliability and resiliency, which may have an adverse affect on performance in certain circumstances.

> Note: The latest version of this tuning guide can be found on the JBossESB wiki.

InVM transport

InVM transport means that the Service can be invoked (via ServiceInvoker) from within the same VM with minimal overhead i.e. without incurring any networking or message serialization overhead. Please note, due to the volatility aspect of the InVM queue, you may not be able to achieve all of the ACID semantics, particularly when used in conjunction with other transactional resources such as databases. For details, please refer to the "InVM Transport" section in Programmers Guide.

The code snippet below shows how to configure your service using InVm transport:

```
        <service category="HelloWorld" name="Service1"
description="Service 1" invmScope="GLOBAL">
            <listeners>
                <!-- So we just need to define a Gateway to the
service... -->
                <jms-listener name="JMS-Gateway"
busidref="quickstartGwChannel" is-gateway="true"/>
            </listeners>
            <actions>
                <action name="println"
class="org.jboss.soa.esb.actions.SystemPrintln">
                    <property name="message" value=" - > Service 1"/
>
                </action>

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

Maximum threads for MessageAwareListener

The default value is 1. Following example shows how to set the max threads number to 100:

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

Maximum threads for jbr listener

The default number is 50. Following example shows how to set the max threads number to 100:

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

Message Filters

Message filters are used to dynamically augment the message for example, adding transaction or security information when the message flows through the ESB. Depending on the particular message filters that have been configured in your ESB, they may have impacts on the performance. For further information regarding message filters please refer to the "Meta-data and Filters" section in Programmers Guide.

Index