

## **ESB Administrators Guide**

**4.2**

# **JBoss Enterprise SOA Platform**



**ISBN:**

**Publication date: February, 2008**

The SOA Platform edition of the JBoss ESB Administrators Guide

---

# ESB Administrators Guide: JBoss Enterprise SOA Platform

Copyright © 2008 Red Hat, Inc.

Copyright © 2008 Red Hat, Inc.. This material may only be distributed subject to the terms and conditions set forth in the Creative Commons Attribution-Noncommercial-Share Alike 3.0 Unported License (which is presently available at <http://creativecommons.org/licenses/by-nc-sa/3.0/>).

Red Hat and the Red Hat "Shadow Man" logo are registered trademarks of Red Hat, Inc. in the United States and other countries.

All other trademarks referenced herein are the property of their respective owners.

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

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

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



---

Preface .....	vii
1. Document Conventions .....	vii
2. We Need Feedback .....	viii
1. Configuration .....	1
1. Standalone server .....	1
2. JBossESB JMS Providers .....	1
2.1. How can I configure a different JMS provider? .....	1
3. Database Configuration .....	4
4. Using a JSR-170 Message Store .....	6
5. Message Tracing .....	7
2. Registry .....	9
3. Default ReplyTo EPR .....	11
4. ServiceBinding Manager .....	13
5. Monitoring and Management .....	15
1. Monitoring and Management Console .....	15
2. Alerts .....	15
6. Hot Deployment .....	17
1. Server mode .....	17
2. Standalone (bootstrap) mode. ....	18
7. Contract Publishing .....	19
1. Overview .....	19
2. "Contract" Application .....	19
3. Publishing a Contract from an Action .....	20
A. Revision History .....	21

---

---

## Preface

# 1. Document Conventions

Certain words in this manual are represented in different fonts, styles, and weights. This highlighting indicates that the word is part of a specific category. The categories include the following:

*Courier font*

Courier font represents commands, file names and paths, and prompts .

When shown as below, it indicates computer output:

```
Desktop      about.html    logs          paulwesterberg.png
Mail         backupfiles   mail          reports
```

**Courier font**

Bold Courier font represents text that you are to type, such as: `service jonas start`

If you have to run a command as root, the root prompt (#) precedes the command:

```
# gconftool-2
```

*italic Courier font*

Italic Courier font represents a variable, such as an installation directory:

```
install_dir/bin/
```

**font**

Bold font represents **application programs** and **text found on a graphical interface**.

When shown like this: **OK** , it indicates a button on a graphical application interface.

Additionally, the manual uses different strategies to draw your attention to pieces of information. In order of how critical the information is to you, these items are marked as follows:



### Note

A note is typically information that you need to understand the behavior of the system.



### Tip

A tip is typically an alternative way of performing a task.



### Important

Important information is necessary, but possibly unexpected, such as a configuration change that will not persist after a reboot.



### Caution

A caution indicates an act that would violate your support agreement, such as recompiling the kernel.



### Warning

A warning indicates potential data loss, as may happen when tuning hardware for maximum performance.

## 2. We Need Feedback

If you find a typographical error in the *ESB Administrators Guide*, or if you have thought of a way to make this manual better, we would love to hear from you! Please submit a report in JIRA: <http://jira.jboss.com/jira> against the Documentation component of the *SOA Platform* project.

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

ESB\_ADG

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



# Configuration

## 1. Standalone server

If you wish to run the JBossESB server on the same machine as JBossAS, then you should refer to the instructions in the SOA Platform Getting Started Guide.

## 2. JBossESB JMS Providers

The JBossESB supports a number of JMS providers. Currently we have successfully tested JBossMQ, ActiveMQ and Websphere MQ Series (version 5.3 and 6.0). For the SOA Platform JBossMQ is the recommended, tested, and supported JMS provider.

### 2.1. How can I configure a different JMS provider?



#### 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.

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 listening to a `destination-type` 'queue' and that its name is 'queue/A'.

1. Each JMSListener and JMSGateway can be configured to use its own JMS provider, so you can use more than 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.

### 2.1.1. 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. Note 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. Please note that JBoss MQ is the supported JMS provider for the SOA Platform.

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

### 2.1.2. 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, e.g., [this page](#)

[[http://labs.jboss.com/file-access/default/members/jbossmessaging/freezone/docs/userguide-1.4.0.SP1/html\\_single/](http://labs.jboss.com/file-access/default/members/jbossmessaging/freezone/docs/userguide-1.4.0.SP1/html_single/)]

### 2.1.3. 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.

### 2.1.4. 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="QUEUEEA"
```

1. Websphere likes all CAPS queue names and no slashes (QUEUEEA), 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` and `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 therefore 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.

### 2.1.5. 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"/>
```

You may notice the reference to the `InitialContext` factory. You only need this is if you want to avoid OracleAQ 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.

### 2.1.6. 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.

## 3. Database Configuration

The ESB uses databases 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 that you will need to change, based on the connection information appropriate to your environment. These settings are found in the `DBSTORE` section of the file.

As long as there is a script for your database the ESB will auto-create the schemas 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"/>
<property name="org.jboss.soa.esb.persistence.db.conn.manager"
<property name="org.jboss.soa.esb.persistence.db.connection.url"
  value="jdbc:hsqldb:hsqldb://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
<code>org.jboss.soa.esb.persistence.db.conn.manager</code>	the db connection manager.

<code>org.jboss.soa.esb.persistence.db.datasource</code>	The data source name (used for JNDI lookup)
<code>org.jboss.soa.esb.persistence.db.connection.url</code>	this is the db connection url for your database.
<code>org.jboss.soa.esb.persistence.db.jdbc.driver</code>	JDBC Driver
<code>org.jboss.soa.esb.persistence.db.user</code>	db user
<code>org.jboss.soa.esb.persistence.db.pwd</code>	db password
<code>org.jboss.soa.esb.persistence.db.pool.initialsize</code>	initial size of db connection pool
<code>org.jboss.soa.esb.persistence.db.pool.minimum</code>	minimum size of db connection pool
<code>org.jboss.soa.esb.persistence.db.pool.maximum</code>	maximum size of db connection pool
<code>org.jboss.soa.esb.persistence.db.pool.testtable</code>	A table name (created dynamically by pool manager) to test for valid connections in the pool
<code>org.jboss.soa.esb.persistence.db.pool.timeout</code>	timeout period to wait for connection requests from pool

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

1. Use of HSQLDB for production is not recommended, and is not a supported configuration for the SOA Platform.

## 4. 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.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"/>
```

1. `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.
2. `jcr.username` - username for getting a repository session
3. `jcr.password` - password for getting a repository session
4. `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.

## 5. 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.message.trace` 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.



# Registry

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

- When run, services typically register the EPR through which they can be contacted. Well behaved services should remove EPRs from the registry when they terminate. However, machine crashes or incorrectly developed services may leave stale entries in the registry that prevent the correct execution of subsequent deployments. In that case those entries may be removed manually. It is important that you ensure the system is in a quiescent state before doing this.
- If you set the optional `remove-old-service` tag 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.



## 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 for 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.

1. 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`
2. 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.
3. 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 application server documentation contains instructions on how to set up the ServiceBinding manager.



### Note

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 :

1. `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.
2. `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.
3. `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.
4. `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.

## 1. Monitoring and Management Console

JBossESB has its own monitoring and management console for ESB related properties (<http://localhost:8080/jbossesb>). The Monitoring and Management Console is covered in-depth in its own help documentation (`docs/governance/Monitoring.pdf`).

## 2. Alerts

The JBoss Web Console (<http://jboss.org/wiki/Wiki.jsp?page=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.

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=
      <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* - when the threshold is reached. If you want to be alerted again after resetting the counter, you need to also 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 <http://wiki.jboss.org/wiki/Wiki.jsp?page=JBossMonitoring>



# Hot Deployment

## 1. Server mode

JBossAS as well as the JBossESB-Server are always checking the `deploy` directory for new files to deploy. There are additional steps to deploy beyond copying a file to that directory. However, on occasion you may need to *redploy* an existing component. Here are the instructions for doing this:

### **sar files.**

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

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

### **esb files.**

Any `*.esb` archive will redeploy when:

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

JBoss ESB actions have lifecycle support, so when hot deployed a component 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) [<http://wiki.jboss.org/wiki/Wiki.jsp?page=JBossESBQuickStart>]).

### **Rule files.**

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

1. redeploy the `jbrules.esb` (see 2)
2. turn on the `ruleReload` in the action config (see [JBossESBContentBasedRouting](http://wiki.jboss.org/wiki/Wiki.jsp?page=JBossESBContentBasedRouting) [<http://wiki.jboss.org/wiki/Wiki.jsp?page=JBossESBContentBasedRouting>]). Once this is done when a rule file *changes* it will be reloaded.

### **Transformation files.**

There are two options to refresh transformation files:

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

### **Business Process Definitions.**

When using jBPM new Business Process Definitions can be deployed. From within the jBBPM eclipse plugin you can deploy a new definition to the jbpn 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.

## **2. Standalone (bootstrap) mode.**

The bootstrapper does not deploy `esb` archives. You can only have one `jboss-esb.xml` configuration file per node. It will monitor the timestamp on this file and it will reread the configuration if a change occurs. To update rules you will have to use the `ruleReload`; to update transformations you need to send out a Smooks JMS notification using the `esb-console`; and finally to update BPDs you can follow the process mentioned in [Business Process Definitions](#).

# Contract Publishing

## 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 the JBoss SOA Platform ESB Message Action Guide).

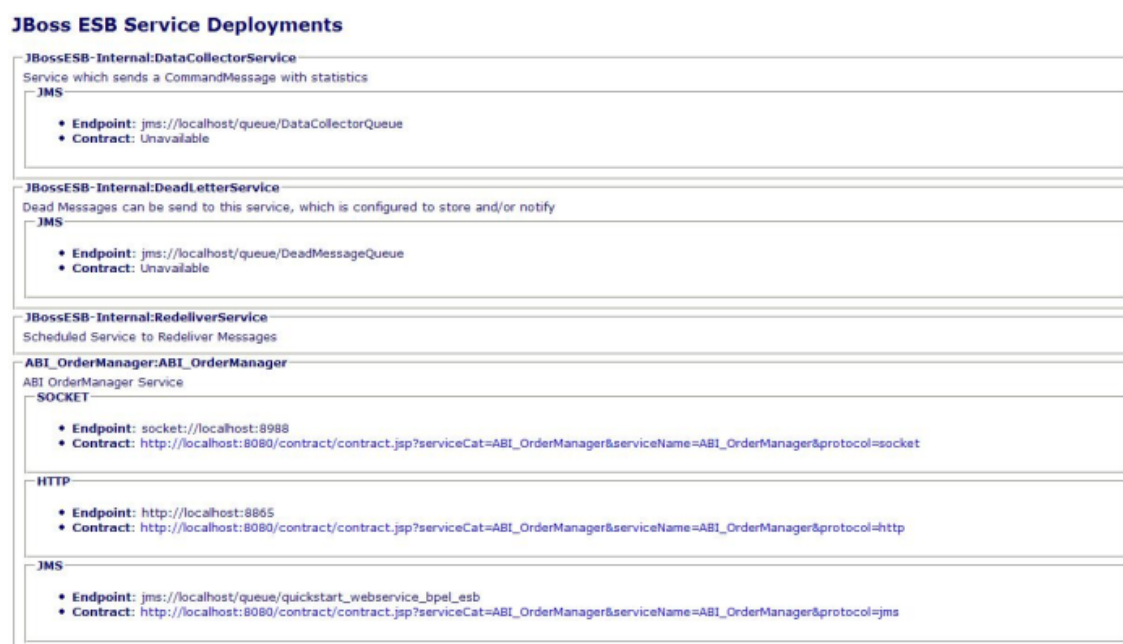
## 2. "Contract" Application

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)<sup>1</sup>.

It can be accessed through the following URL:

<http://localhost:8080/contract/>

The following is a screenshot of this application.



**Figure 7.1. 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

<sup>1</sup> 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.

hyperlink links off to the WSDL.

### 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 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](#)

[<http://anonsvn.labs.jboss.com/labs/jbossesb/trunk/product/services/soap/src/main/java/org/jboss/soa/esb/actions/s>

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

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

[See the `WebServiceContractPublisher` code as an example](#)

[<http://anonsvn.labs.jboss.com/labs/jbossesb/trunk/product/services/soap/src/main/java/org/jboss/soa/esb/actions/s>

---

# Appendix A. Revision History

Revision History

Revision 1.0

