JBoss Enterprise SOA Platform 5

Administration Guide

This guide is for corporate system administrators.

Edition 5.3.1

Last Updated: 2017-10-27

JBoss Enterprise SOA Platform 5 Administration Guide

This guide is for corporate system administrators.
Edition 5.3.1

David Le Sage
Red Hat Engineering Content Services

Suzanne Dorfield
Red Hat Engineering Content Services

Legal Notice

Abstract

This document will guide the user through the day-to-day administration tasks for the JBoss Enterprise SOA Platform

Table of Contents

PREFACE

PART I. THE BASICS

CHAPTER 1. PREFACE

1.1. BUSINESS INTEGRATION

In order to provide a dynamic and competitive business infrastructure, it is crucial to have a service-oriented architecture in place that enables your disparate applications and data sources to communicate with each other with minimum overhead.

The JBoss Enterprise SOA Platform is a framework capable of orchestrating business services without the need to constantly reprogram them to fit changes in business processes. By using its business rules and message transformation and routing capabilities, JBoss Enterprise SOA Platform enables you to manipulate business data from multiple sources.

Report a bug

1.2. WHAT IS A SERVICE-ORIENTED ARCHITECTURE?

Introduction

A *Service Oriented Architecture* (SOA) is not a single program or technology. Think of it, rather, as a software design paradigm.

As you may already know, a *hardware bus* is a physical connector that ties together multiple systems and subsystems. If you use one, instead of having a large number of point-to-point connectors between pairs of systems, you can simply connect each system to the central bus. An *enterprise service bus* (ESB) does exactly the same thing in software.

The ESB sits in the architectural layer above a messaging system. This messaging system facilitates *asynchronous communications* between services through the ESB. In fact, when you are using an ESB, everything is, conceptually, either a *service* (which, in this context, is your application software) or a *message* being sent between services. The services are listed as connection addresses (known as *end-points references*.)

It is important to note that, in this context, a "service" is not necessarily always a web service. Other types of applications, using such transports as File Transfer Protocol and the Java Message Service, can also be "services."

NOTE

At this point, you may be wondering if an enterprise service bus is the same thing as a service-oriented architecture. The answer is, "Not exactly." An ESB does not form a service-oriented architecture of itself. Rather, it provides many of the tools than can be used to build one. In particular, it facilitates the *loose-coupling* and *asynchronous message passing* needed by a SOA. Always think of a SOA as being more than just software: it is a series of principles, patterns and best practices.

Report a bug

1.3. KEY POINTS OF A SERVICE-ORIENTED ARCHITECTURE

These are the key components of a service-oriented architecture:

1. the *messages* being exchanged
2. the *agents* that act as service requesters and providers
3. the *shared transport mechanisms* that allow the messages to flow back and forth.

Report a bug

1.4. WHAT IS THE JBOSS ENTERPRISE SOA PLATFORM?

The JBoss Enterprise SOA Platform is a framework for developing enterprise application integration (EAI) and service-oriented architecture (SOA) solutions. It is made up of an enterprise service bus (JBoss ESB) and some business process automation infrastructure. It allows you to build, deploy, integrate and orchestrate business services.

Report a bug

1.5. THE SERVICE-ORIENTED ARCHITECTURE PARADIGM

The service-oriented architecture (SOA) consists of three roles: requester, provider, and broker.

Service Provider

A service provider allows access to services, creates a description of a service and publishes it to the service broker.

Service Requester

A service requester is responsible for discovering a service by searching through the service descriptions given by the service broker. A requester is also responsible for binding to services provided by the service provider.

Service Broker

A service broker hosts a registry of service descriptions. It is responsible for linking a requester to a service provider.

Report a bug

1.6. CORE AND COMPONENTS

The JBoss Enterprise SOA Platform provides a comprehensive server for your data integration needs. On a basic level, it is capable of updating business rules and routing messages through an Enterprise Service Bus.

The heart of the JBoss Enterprise SOA Platform is the Enterprise Service Bus. JBoss (ESB) creates an environment for sending and receiving messages. It is able to apply “actions” to messages to transform them and route them between services.

There are a number of components that make up the JBoss Enterprise SOA Platform. Along with the ESB, there is a registry (jUDDI), transformation engine (Smooks), message queue (HornetQ) and BPEL engine (Riftsaw).

Report a bug

1.7. COMPONENTS OF THE JBOSS ENTERPRISE SOA PLATFORM

- A full Java EE-compliant application server (the JBoss Enterprise Application Platform)
- an enterprise service bus (JBoss ESB)
- a business process management system (jBPM)
- a business rules engine (JBoss Rules)
- support for the optional JBoss Enterprise Data Services (EDS) product.

Report a bug

1.8. JBOSS ENTERPRISE SOA PLATFORM FEATURES

The JBoss Enterprise Service Bus (ESB)

The ESB sends messages between services and transforms them so that they can be processed by different types of systems.

Business Process Execution Language (BPEL)

You can use web services to orchestrate business rules using this language. It is included with SOA for the simple execution of business process instructions.

Java Universal Description, Discovery and Integration (jUDDI)

This is the default service registry in SOA. It is where all the information pertaining to services on the ESB are stored.

Smooks

This transformation engine can be used in conjunction with SOA to process messages. It can also be used to split messages and send them to the correct destination.

JBoss Rules

This is the rules engine that is packaged with SOA. It can infer data from the messages it receives to determine which actions need to be performed.

Report a bug

1.9. FEATURES OF THE JBOSS ENTERPRISE SOA PLATFORM'S JBOSSESB COMPONENT

The JBoss Enterprise SOA Platform's JBossESB component supports:

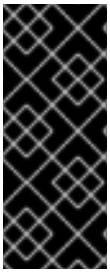
- Multiple transports and protocols
- A listener-action model (so that you can loosely-couple services together)

- Content-based routing (through the JBoss Rules engine, XPath, Regex and Smooks)
- Integration with the JBoss Business Process Manager (jBPM) in order to provide service orchestration functionality
- Integration with JBoss Rules in order to provide business rules development functionality.
- Integration with a BPEL engine.

Furthermore, the ESB allows you to integrate legacy systems in new deployments and have them communicate either synchronously or asynchronously.

In addition, the enterprise service bus provides an infrastructure and set of tools that can:

- Be configured to work with a wide variety of transport mechanisms (such as e-mail and JMS),
- Be used as a general-purpose object repository,
- Allow you to implement pluggable data transformation mechanisms,
- Support logging of interactions.



IMPORTANT

There are two trees within the source code: **`org.jboss.internal.soa.esb`** and **`org.jboss.soa.esb`**. Use the contents of the **`org.jboss.internal.soa.esb`** package sparingly because they are subject to change without notice. By contrast, everything within the **`org.jboss.soa.esb`** package is covered by Red Hat's deprecation policy.

Report a bug

1.10. TASK MANAGEMENT

JBoss SOA simplifies tasks by designating tasks to be performed universally across all systems it affects. This means that the user does not have to configure the task to run separately on each terminal. Users can connect systems easily by using web services.

Businesses can save time and money by using JBoss SOA to delegate their transactions once across their networks instead of multiple times for each machine. This also decreases the chance of errors ocurring.

Report a bug

1.11. INTEGRATION USE CASE

Acme Equity is a large financial service. The company possesses many databases and systems. Some are older, COBOL-based legacy systems and some are databases obtained through the acquisition of smaller companies in recent years. It is challenging and expensive to integrate these databases as business rules frequently change. The company wants to develop a new series of client-facing e-commerce websites, but these may not synchronise well with the existing systems as they currently stand.

The company wants an inexpensive solution but one that will adhere to the strict regulations and security requirements of the financial sector. What the company does not want to do is to have to write and maintain “glue code” to connect their legacy databases and systems.

The JBoss Enterprise SOA Platform was selected as a middleware layer to integrate these legacy systems with the new customer websites. It provides a bridge between front-end and back-end systems. Business rules implemented with the JBoss Enterprise SOA Platform can be updated quickly and easily.

As a result, older systems can now synchronise with newer ones due to the unifying methods of SOA. There are no bottlenecks, even with tens of thousands of transactions per month. Various integration types, such as XML, JMS and FTP, are used to move data between systems. Any one of a number of enterprise-standard messaging systems can be plugged into JBoss Enterprise SOA Platform providing further flexibility.

An additional benefit is that the system can now be scaled upwards easily as more servers and databases are added to the existing infrastructure.

Report a bug

1.12. UTILISING THE JBOSS ENTERPRISE SOA PLATFORM IN A BUSINESS ENVIRONMENT

Cost reduction can be achieved due to the implementation of services that can quickly communicate with each other with less chance of error messages occurring. Through enhanced productivity and sourcing options, ongoing costs can be reduced.

Information and business processes can be shared faster because of the increased connectivity. This is enhanced by web services, which can be used to connect clients easily.

Legacy systems can be used in conjunction with the web services to allow different systems to "speak" the same language. This reduces the amount of upgrades and custom code required to make systems synchronise.

Report a bug

CHAPTER 2. INTRODUCING THE JBOSS ENTERPRISE SOA PLATFORM

2.1. INTENDED AUDIENCE

This book has been written for corporate system administrators undertaking day-to-day administration, configuration and monitoring tasks with the JBoss Enterprise SOA Platform.

Report a bug

2.2. AIM OF THIS BOOK

Read this book in order to learn how to do a myriad of day-to-day tasks to maintain your corporation's installation of the JBoss Enterprise SOA Platform. User administration, logging, auditing, monitoring, security and troubleshooting are all covered in this guide. It is assumed that your system had already been installed and configured correctly as per the instructions in the *Installation and Configuration Guide*.

Report a bug

2.3. BACK UP YOUR DATA

WARNING

Red Hat recommends that you back up your system settings and data before undertaking any of the configuration tasks mentioned in this book.

Report a bug

2.4. RED HAT DOCUMENTATION SITE

Red Hat's official documentation site is at https://access.redhat.com/knowledge/docs/. There you will find the latest version of every book, including this one.

Report a bug

2.5. VARIABLE NAME: SOA_ROOT DIRECTORY

SOA Root (often written as SOA_ROOT) is the term given to the directory that contains the application server files. In the standard version of the JBoss Enterprise SOA Platform package, SOA root is the **`jboss-soa-p-5`** directory. In the Standalone edition, though, it is the **`jboss-soa-p-standalone-5`** directory.

Throughout the documentation, this directory is frequently referred to as **SOA_ROOT**. Substitute either **jboss-soa-p-5** or **jboss-soa-p-standalone-5** as appropriate whenever you see this name.

Report a bug

2.6. VARIABLE NAME: PROFILE

PROFILE can be any one of the server profiles that come with the JBoss Enterprise SOA Platform product: default, production, all, minimal, standard or web. Substitute one of these that you are using whenever you see "PROFILE" in a file path in this documentation.

Report a bug

CHAPTER 3. RUNNING THE JBOSS ENTERPRISE SOA PLATFORM IN A TESTING ENVIRONMENT

3.1. START THE JBOSS ENTERPRISE SOA PLATFORM

Prerequisites

The following software must be installed:

- JBoss Enterprise SOA Platform

Procedure 3.1. Start the JBoss Enterprise SOA Platform

- **Start the SOA server in a *server window***
  - **Red Hat Enterprise Linux**
    a. Open a terminal and navigate to the **bin** directory by entering the command **cd *SOA_ROOT*/jboss-as/bin**.
    b. Enter **./run.sh** to start the SOA server. (Because you are not specifying a server profile, "default" will be used.)
  - **Microsoft Windows**
    a. Open a terminal and navigate to the **bin** directory by entering the command **chdir *SOA_ROOT*\jboss-as\bin**.
    b. Enter **run.bat** to start the SOA server. (Because you are not specifying a server profile, "default" will be used.)

Result

The server starts. Note that this will take approximately two minutes, depending on the speed of your hardware.

NOTE

To verify that there have been no errors, check the server log: **less *SOA_ROOT*/jboss-as/server/*PROFILE*/log/server.log**. As another check, open a web browser and go to http://localhost:8080. Make sure you can login to the admin console with the user name and password you have set.

Report a bug

3.2. DEPLOY THE "HELLO WORLD" QUICKSTART ON YOUR TEST SERVER

Prerequisites

- Check that the setting in **SOA_ROOT/jboss-as/samples/quickstarts/conf/quickstarts.properties-example** matches the server configuration (**default** in a testing environment).

Procedure 3.2. Deploy the "Hello World" Quickstart

1. Check that the server has fully launched.
2. Open a second terminal window and navigate to the directory containing the quick start: **cd SOA_ROOT/jboss-as/samples/quickstarts/helloworld** (or **chdir SOA_ROOT\jboss-as\samples\quickstarts\helloworld** in Microsoft Windows).
3. Run **ant deploy** to deploy the quickstart. Look for messages such as this to confirm if the deployment was successful:

```
deploy-esb:
     [copy] Copying 1 file to
/jboss/local/53_DEV2/jboss-soa-p-5/jboss-as/server/default/deploy

deploy-exploded-esb:

quickstart-specific-deploys:
     [echo] No Quickstart specific deployments being made.

display-instructions:
     [echo]
     [echo] ******************
     [echo] Quickstart deployed to target JBoss ESB/App Server at
'/jboss/local/53_DEV2/jboss-soa-p-5/jboss-as/server/default/deploy'.
     [echo] 1.  Check your ESB Server console to make sure the
deployment was
executed without errors.
     [echo] 2.  Run 'ant runtest' to run the Quickstart.
     [echo] 3.  Check your ESB Server console again.  The Quickstart
should
have produced some output.
     [echo] ******************

deploy:

BUILD SUCCESSFUL
```

Also, check for this in the **SOA_ROOT/jboss-as/server/default/log/server.log**:

```
10:58:52,998 INFO  [NamingHelper] JNDI InitialContext properties:{}
11:00:58,154 INFO  [QueueService]
Queue[/queue/quickstart_helloworld_Request_esb] started,
fullSize=200000,
pageSize=2000, downCacheSize=2000
11:00:58,186 INFO  [QueueService]
Queue[/queue/quickstart_helloworld_Request_gw] started,
fullSize=200000,
pageSize=2000, downCacheSize=2000
11:00:58,427 INFO  [EsbDeployment] Starting ESB Deployment
'Quickstart_helloworld.esb'
```

4. Run the quickstart by issuing this command: **ant runtest**. When the quickstart is run, messages such as this are written to the **SOA_ROOT/jboss-as/server/default/log/server.log**:

```
11:03:02,190 INFO  [STDOUT]
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
11:03:02,191 INFO  [STDOUT] Body: Hello World
11:03:02,191 INFO  [STDOUT]
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
11:03:02,192 INFO  [STDOUT] Message structure:
11:03:02,192 INFO  [STDOUT] [Hello World].
```

Result

The words "Hello World" will appear on the server terminal. This message will also be appended to the **SOA_ROOT/jboss-as/server/default/log/server.log** file.

Report a bug

3.3. UNDEPLOY THE "HELLO WORLD" QUICKSTART

Procedure 3.3. Task

1. Navigate to the quickstart's directory: **cd SOA_ROOT/jboss-as/samples/quickstarts/helloworld** (or **chdir SOA_ROOT\jboss-as\samples\quickstarts\helloworld** if you are running Microsoft Windows).

2. Run the **ant undeploy** command. You should see messages such as this displayed:

```
undeploy:
[delete] Deleting:
/jboss/local/53_DEV2/jboss-soa-p-5/jboss-
as/server/default/deploy/Quickstart_helloworld.esb

BUILD SUCCESSFUL
```

   And messages such as this written to the server.log:

```
11:10:08,205 INFO  [EsbDeployment] Stopping
'Quickstart_helloworld.esb'
11:10:08,577 INFO  [EsbDeployment] Destroying
'Quickstart_helloworld.esb'
```

Report a bug

3.4. STOP THE JBOSS ENTERPRISE SOA PLATFORM SERVER

Procedure 3.4. Stop the JBoss Enterprise SOA Platform Server

- **Stop the SOA server**
  Press **ctrl-c** in the *server window* (the terminal window where the SOA server was started).

Result

The server will shut down. Note that this process will take a few minutes. Look for this line in the **server.log** file to confirm that the server has shut down successfully:

```
        12:17:02,786 INFO  [ServerImpl] Shutdown complete
```

Report a bug

CHAPTER 4. QUICKSTARTS

4.1. QUICKSTART

The quickstarts are sample projects. Each one demonstrates how to use a specific piece of functionality in order to aid you in building services. There are several dozen quickstarts included in the **`SOA_ROOT/jboss-as/samples/quickstarts/`** directory. Build and deploy every quickstart by using **Apache Ant**.

Report a bug

4.2. IMPORTANT NOTES ABOUT QUICKSTARTS

When intending to run a quickstart, remember the following points:

1. Each quickstart needs to be built and deployed using Apache Ant.
2. Each quickstart uses the **`samples/quickstarts/conf/quickstarts.properties`** file to store environment-specific configuration options such as the directory where the server was installed. You must create a **`quickstarts.properties`** file that matches your server installation. An example properties file (**`quickstarts.properties-example`**) is included.
3. Each quickstart has different requirements. These are documented in their individual **`readme.txt`** files.
4. Not every quickstart can run under every server profile.
5. The jBPM quickstarts require a valid jBPM Console user name and password. Supply these by adding them as properties in the **`SOA_ROOT/jboss-as/samples/quickstarts/conf/quickstarts.properties`** file:

   ```
   # jBPM console security credentials
   jbpm.console.username=admin
   jbpm.console.password=adminpassword
   ```

   The quickstarts that are affected by this requirement are **`bpm_orchestration1`**, **`bpm_orchestration2`**, **`bpm_orchestration3`** and **`bpm_orchestration4`**.
6. You can only execute some of the quickstarts (such as **groovy_gateway**) if the server is not running in *headless* mode. (The JBoss Enterprise SOA Platform is configured to launch in headless mode by default.)

IMPORTANT

Red Hat recommends that you run production servers in headless mode only.

Report a bug

4.3. LEARN MORE ABOUT A QUICKSTART

To learn more about a particular quickstart:

Procedure 4.1. Task

1. Study the quickstart's **`readme.txt`** file.
2. Run the **`ant help`** command in the quickstart's directory.

Report a bug

4.4. OVERVIEW OF HOW THE "HELLO WORLD" QUICKSTART WORKS

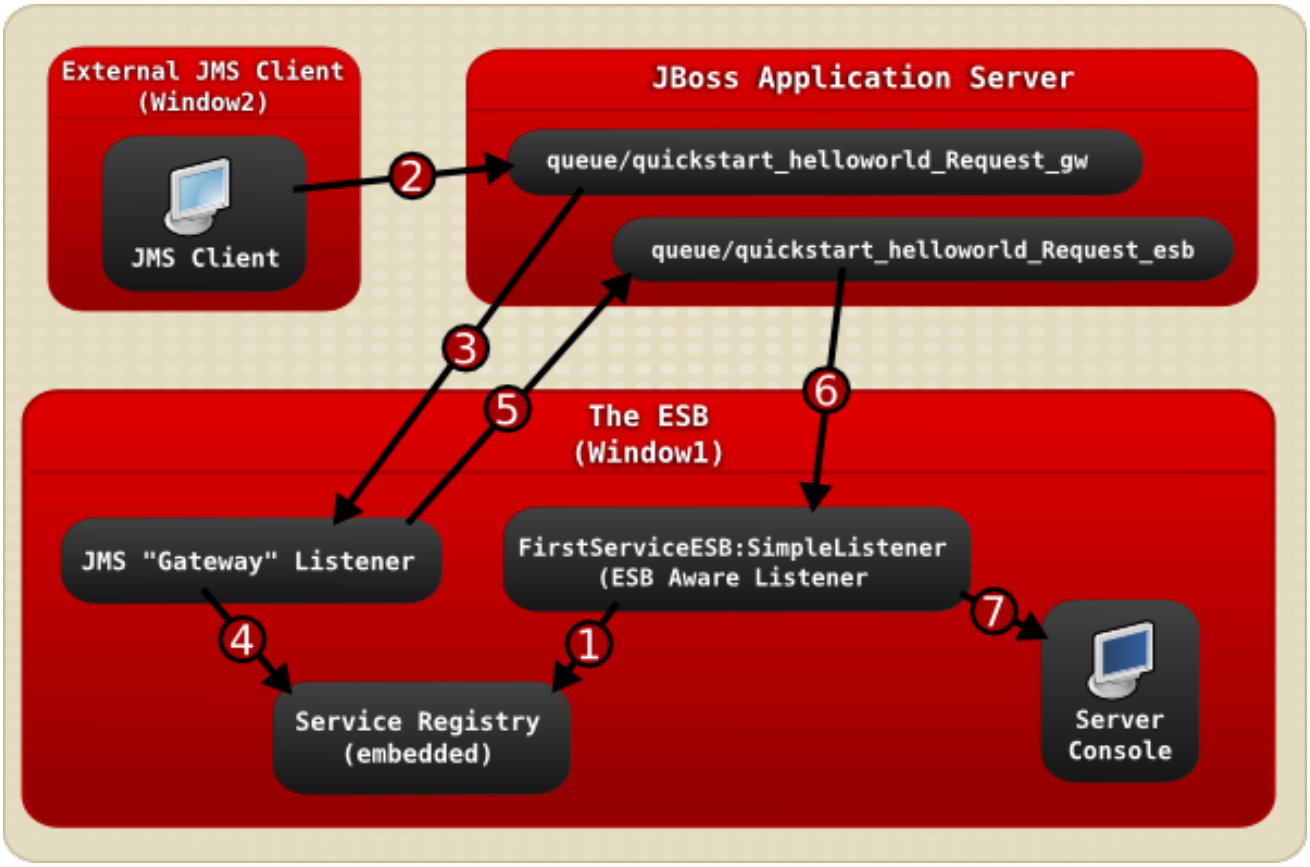


Figure 4.1. Image

1. The **JBoss Enterprise SOA Platform** server is launched in **`Window1`** and then the **`FirstServiceESB:SimpleListener`** service is added to the Service Registry service when the helloworld quickstart is deployed.
2. A JMS client sends an ESB-unaware "Hello World" message, (it is a plain **`String`** object), to the JMS Queue (**`queue/quickstart_helloworld_Request_gw`**).
3. The JMS Gateway Listener receives the ESB-unaware message and creates from it an ESB-aware message for use by ESB-aware end-points.
4. The **`JMS Gateway Listener`** uses the **`service registry`** to find the **`FirstServiceESB:SimpleListener`** service's *end-point reference* (EPR). In this case, the EPR is the **`queue/quickstart_helloworld_Request_esb`** JMS queue.

5. The **`JMS Gateway Listener`** takes the new ESB-aware message and sends it to the **`queue/quickstart_helloworld_Request_esb`** JMS queue.

6. The **`FirstServiceESB:SimpleListener`** service receives the message.

7. The **`FirstServiceESB:SimpleListener`** service extracts the payload from the message and outputs it to the console.

Report a bug

CHAPTER 5. RUNNING THE JBOSS ENTERPRISE SOA PLATFORM IN A PRODUCTION ENVIRONMENT

5.1. SERVER PROFILES

Table 5.1. Server Profiles

Profile	Description
default	Use this profile for development and testing. This profile uses less memory than the production profile but clustering is not enabled in this mode. In addition, this profile provides more verbose logging than the "all" and "production" profiles. This verbose logging provides you with additional information, but adversely affects server performance. Unless you explicitly specify a different profile, this profile is used when the server is started.
production	Use this profile on production servers. This profile provides clustering and maximizes performance by using more memory and providing less verbose logging and screen console output than the "all" or "default" profiles. Note that output (such as the message from the "Hello World" quick start) does not appear on the console screen in this mode. It is written to the log only.
minimal	Enables the minimum features needed for a functioning system. No archives are deployed. No ESB or SOA features are enabled. The BPEL Engine is not available.
standard	This provides standard functionality for testing. No web, ESB, or SOA features are enabled. The BPEL Engine is not available.
web	The jbossweb.sar archives are deployed when this profile is run. No ESB, or SOA features are enabled. The BPEL Engine is not available.
all	All of the pre-packaged ESB archives are deployed when this profile is run. This profile offers less performance and scalability than the "production" profile, but requires less memory to run.

Report a bug

5.2. RUN.SH OPTIONAL SWITCHES

Table 5.2. ./run.sh Optional Switches

Switch	Purpose	Example of Use
-c	Make the server use a specific profile. If none is specified, "default" is used.	`./run.sh -c production`
-b	Bind the server to a specific IP address. If none is specified, the default (127.0.0.1) is used.	`./run.sh -b 10.34.5.2`

Report a bug

5.3. START THE JBOSS ENTERPRISE SOA PLATFORM IN A PRODUCTION ENVIRONMENT

Procedure 5.1. Start the JBoss Enterprise SOA Platform in a Production Environment

1. **Navigate to the bin Directory**
   Open a terminal and input this command: **`cd SOA_ROOT/jboss-as/bin`** (or **`chdir SOA_Root\jboss-as\bin`** in Microsoft Windows).

   NOTE

   It is required that you have set up an administration username and password before proceeding.

2. **Launch the JBoss Enterprise SOA Server on Red Hat Enterprise Linux**
   To start the product, run this command: **`./run.sh -c production`**

3. **Launch the JBoss Enterprise SOA Server on Microsoft Windows**
   To start the product, run this command: **`run.bat -c production`**

Result

The server starts. Note that this may up to around two minutes, depending on the speed of your hardware.

NOTE

To verify that there have been no errors, check the server log: **`less SOA_ROOT/jboss-as/server/PROFILE/log/server.log`**. As another check, open a web browser and go to http://localhost:8080. Make sure you can log into the admin console with the username and password you have set.

Report a bug

5.4. SERVER INSTALLATION

A server installation is a way of configuring the JBoss Enterprise SOA Platform on your system. When the software is installed in this way, it can be launched and shutdown with the host operating system. It is set up just like any other service (or daemon in Linux/Unix terminology) on your operating system.

Report a bug

5.5. CONFIGURE THE JBOSS ENTERPRISE SOA PLATFORM TO RUN AS A RED HAT ENTERPRISE LINUX DAEMON

Procedure 5.2. Task

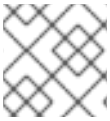
- To make the JBoss Enterprise SOA Platform run as a background daemon (service), you will have to create your own shell script. Red Hat does not supply any scripts to do this.

Report a bug

5.6. START A SERVER INSTALLATION

Prerequisites

- The JBoss Enterprise SOA Platform must be pre-configured to run as a service.



NOTE

This example assumes the service was installed using the name jboss_soa

Procedure 5.3. Task

- To start the JBoss Enterprise SOA Platform as a service, issue this command: **`service jboss_soa start`**

NOTE

If the JBoss user was created as a system account (using the **`-R`** switch) then a warning message is displayed. You can safely ignore this.

Report a bug

5.7. STOP A SERVER INSTALLATION

This example assumes the service was installed using the name jboss_soa

Procedure 5.4. Task

- To stop the JBoss Enterprise SOA Platform when it is running as a service, issue this command: **`service jboss_soa stop`**

Report a bug

PART II. SECURITY

CHAPTER 6. MANAGING USER ACCOUNTS

6.1. USER ACCOUNTS

A user needs an account to be able to log in and use the JBoss Enterprise SOA Platform's various web-based consoles. The default security system reads plain text files (namely **soa-users.properties** and **soa-roles.properties**) to check a user's password and determine their level of access. SOA uses the Java Authentication and Authorization Service (JAAS) to authenticate user accounts.

WARNING

Red Hat does not recommend that you run production servers configured with user passwords in clear text files as it compromises security.

Report a bug

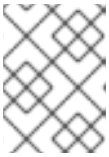
6.2. CREATE USER ACCOUNTS

Procedure 6.1. Add a New User

1. Open the **soa-users.properties** file in a text editor: **vi SOA_ROOT/jboss-as/server/PROFILE/conf/props/soa-users.properties**. Add the user's name and password on a new line, using this syntax: **username=password**.

   Here is an example for a user with the login name "Harold":

   ```
   harold=@dm1nU53r
   ```

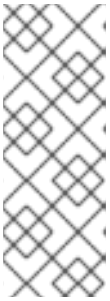
   NOTE

   Any line in this file that begins with a hash (#) is ignored. (You can use this convention to temporarily disable a user account.)

2. Save the changes to the file and exit the text editor.

3. Open the **soa-roles.properties** file in a text editor: **vi SOA_ROOT/jboss-as/server/PROFILE/conf/props/soa-roles.properties**. Add the user and the roles you wish to assign to them on a new line, using this syntax: **username=role1,role2,role3**.

   ```
   harold=JBossAdmin,HttpInvoker,user,admin
   ```



NOTE

You can assign any number of roles. Note that a user must be assigned the **JBossAdmin**, **HttpInvoker**, **user** and **admin** roles in order to be able to log into the server consoles.

Any line in this file that begins with a hash (#) is ignored. You can use this convention to temporarily disable user roles.

4. Save the changes to the file and exit the text editor.

Result

The user will now be able to log in to the server console at http://localhost:8080. You do not have to restart the server.

Report a bug

6.3. SOA-USERS.PROPERTIES

The **soa-users.properties** file is where the user accounts and passwords for accessing the SOA Web consoles are stored. Administrators control access to the system by editing this file. Note that the passwords are saved in clear text so for production systems, password encryption should be used instead.

Report a bug

6.4. SOA-ROLES.PROPERTIES

The **soa-roles.properties** file is where user access privileges are defined. This file uses the following syntax: **username=role1,role2,role3**. You can assign any number of roles. Note that a user must be assigned the **JBossAdmin**, **HttpInvoker**, **user**, and **admin** roles in order to be able to log into the server consoles.

Report a bug

6.5. SECURITY ROLES

Table 6.1. List of Security Permissions for JBoss Enterprise SOA Platform Console Users

Role	Description
JBossAdmin	The JBossAdmin role is required to log into the various management components of SOA. It is the primary role so all system administrators should be assigned this role.
HttpInvoker	The HttpInvoker role is used by the Http Invoker to access JNDIs and EJBs from remote locations.

Role	Description
user	This is used to grant user access to services deployed in SOA if they are configured to utilize the JAAS security domains. The jBPM Console relies on this one role only.
admin	This is used to grant administrative access to services deployed in SOA if they are configured to utilize the JAAS security domains.

Report a bug

6.6. DISABLE A USER'S ACCOUNT

Procedure 6.2. Disable a User's Account

1. Open the **soa-users.properties** file in a text editor: **vi SOA_ROOT/jboss-as/server/PROFILE/conf/props/soa-users.properties**. Either delete the entire line containing the user's name and password or simply put a hash (#) in front of it to "comment it out."

   Here is an example for a user with the login name "Harold":

   ```
   #harold=@dm1nU53r
   ```

2. Save the changes to the file and exit the text editor.

Result

The user will no longer be able to log in to the server console. You do not have to restart the server.

Report a bug

6.7. SECURITY ASSERTION MARKUP LANGUAGE (SAML)

Security Assertion Markup Language (SAML) is a framework compiled in XML. It is used to sercurely pass information between services. It is most commonly used for authentication and identification. With SAML, users can login once and have their identity verified by SAML, thus negating the need to re-enter their credentials repeatedly.

The JBoss ESB uses the SAML provided by PicketLink Project via JAAS Login Modules. It provides users with the ability to assign and validate SAML security tokens.

Report a bug

6.8. ISSUING A SAML SECURITY TOKEN

Procedure 6.3. Task

1. Obtain the Login Module (LM) located in org.picketlink.identity.federation.core.wstrust.auth.STSIssuingLoginModule

2. Open the LM's configuration file.

3. Enter the following code, inserting the names of the services you wish to use:

```
<application-policy name="saml-issue-token">
     <authentication>
         <login-module
code="org.picketlink.identity.federation.core.wstrust.auth.STSIssuin
gLoginModule" flag="required">
             <module-option name="configFile">picketlink-sts-
client.properties</module-option>
             <module-option
name="endpointURI">http://security_saml/goodbyeworld</module-option>
         </login-module>
         <login-module
code="org.picketlink.identity.federation.core.wstrust.auth.STSValida
tingLoginModule" flag="required">
             <module-option name="configFile">picketlink-sts-
client.properties</module-option>
         </login-module>
     </authentication>
 </application-policy>
```

   This configuration uses a stacked LM. The security token from the first LM is later used by the second LM which will validate the security token. Having two separate LMs for this can be useful as there can be situations where you only need to validate a security token.

4. Specify the *picketlink-sts-client* properties:

```
serviceName=PicketLinkSTS
portName=PicketLinkSTSPort
endpointAddress=http://localhost:8080/picketlink-sts/PicketLinkSTS
username=admin
password=admin
```

NOTE

The username and password in this file are only used by the *STSValidatingLoginModule*. The username and password may also be stacked or provided by a callback.

5. To use this LM in JBossESB you need to update your server's *login-config.xml* with the above application-policy. You must also point the ESB service to where you want this LM to be used.

   For example, this is how you could configure it in *jboss-esb.xml*:

```
<service category="SamlSecurityQuickstart" name="issueTokenService"
invmScope="GLOBAL"
    description="This service demonstrates how a service can be
configured to issue and validate a security token">
```

```
    <security moduleName="saml-issue-token"
callbackHandler="org.jboss.soa.esb.services.security.auth.login.JBos
sSTSIssueCallbackHandler">
        <!-- disable the security context timeout so that our
security context is re-evaluated -->
        <property
name="org.jboss.soa.esb.services.security.contextTimeout"
value="0"/>
    </security>
    ...
</service>
```

The callbackHandler that is is specified is specific to the ESB. This is because it requires access to the authentication request in the ESB for retreiving the username and password of the user for whom a security token should be issued.

Report a bug

6.9. VALIDATING A SAML SECURITY TOKEN

Procedure 6.4. Task

1. Open the Login Module (LM) from *org.picketlink.identity.federation.core.wstrust.auth.STSIssuingLoginModule*.
2. Configure the properties file as shown in the example below:

```
        <application-policy name="saml-validate-token">
    <authentication>
        <login-module
code="org.picketlink.identity.federation.core.wstrust.auth.STSValida
tingLoginModule" flag="required">
            <module-option name="configFile">picketlink-sts-
client.properties</module-option>
        </login-module>
    </authentication>
</application-policy>



And in jboss-esb.xml:
<service category="SamlSecurityQuickstart" name="securedSamlService"
invmScope="GLOBAL"
    description="This service demonstrates that an ESB service can
be configured to only validate a security token.">

    <security moduleName="saml-validate-token"
callbackHandler="org.jboss.soa.esb.services.security.auth.login.JBos
sSTSTokenCallbackHandler">
        <!-- disable the security context timeout so that our
security context is re-evaluated -->
```

```
        <property
name="org.jboss.soa.esb.services.security.contextTimeout"
value="0"/>
    </security>
    ...
</service>
```

NOTE

The callbackHandler that is specified is specific to the ESB. This is because it requires access to the authentication request in the ESB for retreiving the SAML Token which is to be validated.

NOTE

An example of SAML support in JBossESB can be found in the security_saml quickstart. More information about the Login Modules provied by PicketLink can be found at http://www.jboss.org/community/wiki/STSLoginModules

Report a bug

6.10. PICKETLINK

The *PicketLink* IDM is a framework that links security systems together. It does so by using identity management. It contains information pertaining to user identification, groups, and permissions.

Report a bug

6.11. INTEGRATION BETWEEN SAML AND PICKETLINK

- The client must first obtain the SAML assertion from PicketLink STS by sending a WS-Trust request to the token service. This process usually involves authentication of the client.
- After obtaining the SAML assertion from the STS, the client includes the assertion in the security context of the EJB request before invoking an operation on the bean.
- Upon receiving the invocation, the EJB container extracts the assertion and validates it by sending a WS-Trust message to the STS. If the assertion is deemed valid by the STS (and the proof of possession token has been verified if needed), the client is authenticated.
- In JBoss, the SAML assertion validation process is handled by the SAML2STSLoginModule. It reads properties from a configurable file (specified by the configFile option) and establishes communication with the STS based on these properties.
- If the assertion is valid, a Principal is created using the assertion subject name. If the assertion contains roles, these roles are also extracted and associated with the caller's Subject.

Report a bug

CHAPTER 7. SECURING YOUR SYSTEM

7.1. SECURING YOUR JBOSS ENTERPRISE SOA PLATFORM INSTALLATION

Introduction

The JBoss Enterprise SOA Platform can be made secure, in the sense that you can configure the product so that services will only be executed if caller authentication succeeds and said caller possesses the correct permissions. The default security implementation is based on JAAS.

There are two ways in which to invoke a service:

1. through a gateway
2. directly via the ServiceInvoker.

When you use the gateway option, it is made responsible for obtaining the security information needed to authenticate the caller. It does this by extracting the needed information from the transport. Once it has done so, it creates an authentication request that is encrypted and passed to the Enterprise Service Bus.

If you use the ServiceInvoker instead, it becomes the client application's responsibility to make the authentication request prior to invoking the service. This entails extracting either the **`UsernameToken`** or the **`BinarySecurityToken`** from the SOAP header's security element.

Report a bug

7.2. JAVA AUTHENTICATION AND AUTHORIZATION SERVICE (JAAS)

The JAAS 1.0 API consists of a set of Java packages designed for user authentication and authorization. The API implements a Java version of the standard Pluggable Authentication Modules (PAM) framework and extends the Java 2 Platform access control architecture to support user-based authorization.

JAAS was first released as an extension package for JDK 1.3 and is bundled with JDK 1.6.

Report a bug

7.3. JAASSECURITYSERVICE

JaasSecurityService is the default implementation of JAAS used in the JBoss Enterprise SOA Platform.

Report a bug

7.4. SECURE YOUR SYSTEM

Procedure 7.1. Task

Open the global configuration file in a text editor: **`vi SOA_ROOT/jboss-as/server/PROFILE/deployers/esb.deployer/jbossesb-properties.xml`**.

1. Scroll down to the section that contains properties name="security" and edit the settings to suit your system:

```
<properties name="security">
<property
name="org.jboss.soa.esb.services.security.implementationClass"
value="org.jboss.internal.soa.esb.services.security.JaasSecurityServ
ice"/>

<property name="org.jboss.soa.esb.services.security.callbackHandler"
value=
"org.jboss.internal.soa.esb.services.security.UserPassCallbackHandle
r"/>

<property name="org.jboss.soa.esb.services.security.sealAlgorithm"
value="TripleDES"/>

<property name="org.jboss.soa.esb.services.security.sealKeySize"
value="168"/>

<property name="org.jboss.soa.esb.services.security.contextTimeout"
value="30000"/>

<property name=
"org.jboss.soa.esb.services.security.contextPropagatorImplemtationCl
ass"
value=
"org.jboss.internal.soa.esb.services.security.JBossASContextPropagat
or"/>

<property name="org.jboss.soa.esb.services.security.publicKeystore"
value="/publicKeyStore"/>


<property
name="org.jboss.soa.esb.services.security.publicKeystorePassword"
value="testKeystorePassword"/>

<property name="org.jboss.soa.esb.services.security.publicKeyAlias"
value="testAlias"/>

<property
name="org.jboss.soa.esb.services.security.publicKeyPassword"
value="testPassword"/>

<property
name="org.jboss.soa.esb.services.security.publicKeyTransformation"
value="RSA/ECB/PKCS1Padding"/>

</properties>
```

2. Save the file and exit.

3. Open the log-in configuration file in your text editor: **vi SOA_ROOT/server/PROFILE/conf/login-config.xml**

4. Configure the JAAS log-in modules by editing the settings in this file. (You can use either a pre-configured option or create your own custom solution.)

5. Save the file and exit.

Report a bug

7.5. CREATE AN ENCRYPTED PASSWORD FILE

Procedure 7.2. Task

1. Go to the **conf** directory: **cd SOA_ROOT/jboss-as/server/PROFILE/conf**

2. Execute this command: **java -cp ../../../lib/jbosssx.jar org.jboss.security.plugins.FilePassword welcometojboss 13 testpass esb.password**

Result

An encrypted password file is created.

Report a bug

7.6. ENCRYPTION OPTIONS

Table 7.1. Encryption Options

Option	Description
Salt	This is the "salt" used to encrypt the password file. (In the example above, it is the **welcometojboss** string .)
Iteration	This is the number of iterations. (In the example above, it is the number **13**.)
Password File Name	This is the name of the file where the encrypted password will be saved. In the example above, it is the **esb.password** string.
testpass	This is the test password.

Report a bug

7.7. CLEAR-TEXT PASSWORD

A clear-text password is the plain text version of a password. It has either not been encrypted or has just been decrypted. Clear text passwords are unsecure.

Report a bug

7.8. PASSWORD MASK

A password mask is a template which determines what characters are allowed to be used in a password. For example, some password masks dictate that a password can only be alphanumeric while others allow special characters like ! and $ signs. Passwords which contain special characters are generally viewed as being more secure.

Report a bug

7.9. MASKING PASSWORDS

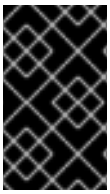
Introduction

Passwords are secret authentication tokens that are used to limit access to resources to authorized parties only. For a JBoss services to access password-protected resources, the password must obviously be made available to it.

This can be done by means of command line arguments passed to the JBoss Enterprise SOA Platform on launch, however this is not practical in a production environment. Instead, passwords are normally made available to JBoss services through their inclusion in configuration files.

All JBoss Enterprise SOA Platform configuration files should be stored on secure file systems, and be made readable by the process owner only.

For an added level of security, you can also mask the password in the configuration file. This section will tell you how to do so.



IMPORTANT

There is no such thing as impenetrable security. Masking passwords is no exception - it is not impenetrable, but it does defeat casual inspection of configuration files, and increases the amount of effort that will be required to extract the password.

Report a bug

7.10. MASK A CLEAR-TEXT PASSWORD

IMPORTANT

You should only perform this key store password encryption procedure once. If you make a mistake entering the keystore password, or you change the key store at a later date, you should delete the **`jboss-keystore_pass.dat`** file and repeat the procedure. Be aware that if you change the key store any masked passwords that were previously generated will no longer function.

Procedure 7.3. Task

1. Generate a key pair using this command: **`keytool -genkey -alias jboss -keyalg RSA -keysize 1024 -keystore password.keystore`** and follow the prompts:

```
        keytool -genkey -alias jboss -keyalg RSA -keysize 1024 -
keystore password.keystore
Enter keystore password:
Re-enter new password:
What is your first and last name?
  [Unknown]:  Bob Bobson
What is the name of your organizational unit?
  [Unknown]:  Corporate_IT
What is the name of your organization?
  [Unknown]:  XYZ
What is the name of your City or Locality?
  [Unknown]:  BRISBANE
What is the name of your State or Province?
  [Unknown]:  QLD
What is the two-letter country code for this unit?
  [Unknown]:  AU
Is CN=Bob Bobson, OU=Corporate_IT, O=XYZ, L=BRISBANE, ST=QLD, C=AU
correct?
  [no]:  yes

Enter key password for jboss
 (RETURN if same as keystore password):
```

NOTE

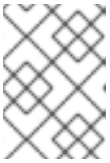
You must specify the same password for the key store and key pair.

2. Run **chown** to change ownership to the JBoss Application Server process owner, and **chmod 600 password.keystore** to make sure only the file's owner can read it.

NOTE

The process owner should not have console log-in access. In that case you will be performing these operations as another user. Creating masked passwords requires read access to the key store, so you may wish to complete configuration of masked passwords before restricting the key store file permissions.

3. Navigate to the **jboss-as/bin** directory: **cd SOA_ROOT/jboss-as/bin**
4. Run the password tool, using the command **./password_tool.sh** on Red Hat Enterprise Linux systems, (or **password_tool.bat** on Microsoft Windows-based systems.)
5. Select **0: Encrypt Keystore Password** by pressing 0, then Enter.
6. Enter the key store password you specified above.
7. Enter a random string of characters to aid with encryption strength. This is the salt.
8. Enter a whole number for the iterator count to aid with encryption strength.
9. Select **5: Exit** to exit.



NOTE

The password tool will exit with the message: **`Keystore is null. Cannot store.`** This is normal.

10. Use the **`chown`** command to change ownership of the **`password/jboss_keystore_pass.dat`** file to the process owner, and **`chmod 600 jboss-keystore_pass.dat`** to ensure that only that owner can read the file.

11. Navigate to the **`jboss-as/bin`** directory: **`cd SOA_ROOT/jboss-as/bin`**

12. Run the password tool, using the command **`./password_tool.sh`** on Red Hat Enterprise Linux systems (or **`password_tool.bat`** on Microsoft Windows systems).

13. Select **`1: Specify KeyStore`** by pressing 1 then Enter.

14. Enter the path to the key store you created above. (You can specify an absolute path, or the path relative to **`SOA_ROOT/jboss-as/bin`**. This should be **`SOA_ROOT/jboss-as/bin/password.keystore`**, unless you have changed the defaults.)

15. Enter the key alias. This should be "jboss" (unless you have performed an advanced installation and changed the defaults).

16. Select **`2: Create Password`** by pressing 2, then Enter. You will be prompted for the security domain. Follow the prompts on screen.

```
        /password_tool.sh
*********************************
****  JBoss Password Tool********
*********************************
Error while trying to load data:Encrypted password file not located
Maybe it does not exist and need to be created.
0: Encrypt Keystore Password 1:Specify KeyStore 2:Create Password
3: Remove a domain 4:Enquire Domain 5:Exit
1
Enter Keystore location including the file name
password.keystore
Enter Keystore alias
jboss
0: Encrypt Keystore Password 1:Specify KeyStore 2:Create Password
3: Remove a domain 4:Enquire Domain 5:Exit
2
Enter security domain:

default
Enter passwd:
passwordmask
Password created for domain:default
0: Encrypt Keystore Password 1:Specify KeyStore 2:Create Password
3: Remove a domain 4:Enquire Domain 5:Exit
```

17. Enter a name for the password mask. This is an arbitrary unique name that you will use to identify the password mask in configuration files.

18. Enter the password that you wish to mask.

19. Repeat the password mask creation process to create masks for all passwords you wish to mask.
20. Exit the program by choosing **`5: Exit`**
21. Navigate to the **`password`** directory: **`cd SOA_ROOT/jboss-as/bin/password`**

Report a bug

7.11. REPLACE A CLEAR TEXT PASSWORD WITH ITS PASSWORD MASK

Prerequisites

- Pre-existing password masks

Procedure 7.4. Task

- Launch a text editor and replace each occurrence of a clear text password in the configuration files with an annotation referencing its mask.

  This is the general form of the annotation:

```
<annotation>
@org.jboss.security.integration.password.Password(securityDomain=MAS
K_NAME, methodName=setPROPERTY_NAME)
</annotation>
```

  As a concrete example, the JBoss Messaging password is stored in the server profile's **`deploy/messaging/messaging-jboss-beans.xml`** file. If you create a password mask named "messaging", then the before and after snippet of the configuration file will looks like this:

```
<property name="suckerPassword">
CHANGE ME!!
</property>
```

```
<annotation>
@org.jboss.security.integration.password.Password(securityDomain=mes
saging,
methodName=setSuckerPassword)
</annotation>
```

Report a bug

7.12. CHANGE THE DEFAULT PASSWORD MASK SETTINGS

By default the server profiles are configured to use the keystore **`SOA_ROOT/jboss-as/bin/password/password.keystore`**, and the key alias "jboss". If you store the key pair used for password masking elsewhere, or under a different alias, you will need to update the server profiles with the new location or key alias.

Procedure 7.5. Task

1. Open the security configuration file in a text editor: **`vi SOA_ROOT/jboss-as/server/PROFILE/deploy/security/security-jboss-beans.xml`**.
2. Edit the key store location and key alias. Here are some example settings:

```
<!-- Password Mask Management Bean-->
   <bean name="JBossSecurityPasswordMaskManagement"

class="org.jboss.security.integration.password.PasswordMaskManagement" >
         <property
name="keyStoreLocation">password/password.keystore</property>
         <property name="keyStoreAlias">jboss</property>
         <property
name="passwordEncryptedFileName">password/jboss_password_enc.dat</property>
         <property
name="keyStorePasswordEncryptedFileName">password/jboss_keystore_pass.dat</property>
   </bean>
```

3. Save the file and exit.

Report a bug

7.13. GLOBAL CONFIGURATION FILE SECURITY SETTINGS

Table 7.2. `jbossesb-properties.xml` Security Settings

Property	Description	Required?
`org.jboss.soa.esb.services.security.implementationClass`	This is the "concrete"*SecurityService* implementation that should be used. The default setting is **`JaasSecurityService`**.	Yes
`org.jboss.soa.esb.services.security.callbackHandler`	This is a default **`CallbackHandler`** implementation, utilized when a JAAS-based **`SecurityService`** is employed. See “Customizing Security” for more information about the CallbackHandler property.	No
`org.jboss.soa.esb.services.security.sealAlgorithm`	This is the algorithm to use when "sealing" the **`SecurityContext`**.	No

Property	Description	Required?
org.jboss.soa.esb.services.security.sealKeySize	This is the size of the secret/symmetric key used to encrypt/decrypt the **SecurityContext**.	No
org.jboss.soa.esb.services.security.contextTimeout	This is the amount of time (in milliseconds) for which a security context is valid. A global setting, this may be over-ridden on a per-service basis. To do so, specify the property of the same name that exists on the security element in the **jboss-esb.xml** file.	No
org.jboss.soa.esb.services.security.contextPropagatorImplementationClass	Use this to configure a global **SecurityContextPropagator**. (For more details on the **SecurityContextPropagator**, please refer to the section on "Advanced Security Options".)	No
org.jboss.soa.esb.services.security.publicKeystore	This is the *Keystore* which holds the keys used to encrypt and decrypt that data which is external to the Enterprise Service Bus. The Keystore is used to encrypt the **AuthenticationRequest**.	No
org.jboss.soa.esb.services.security.publicKeystorePassword	This is the password for the public keystore.	No
org.jboss.soa.esb.services.security.publicKeyAlias	This is the alias to use for the public key.	No
org.jboss.soa.esb.services.security.publicKeyPassword	This is the password for the alias if one was specified upon creation.	No
org.jboss.soa.esb.services.security.publicKeyPassword	This is a cipher transformation. It is in this format: **algorithm/mode/padding**. If this is not specified, the "keys" algorithm will be used by default.	No

Report a bug

7.14. KEY PAIR

A key pair is a set of security tools consisting of a public key and a private key. The public key is used for encryption and the private key is used for decryption.

Report a bug

7.15. KEYSTORE

A keystore is a security mechanism. It contains a number of security certificates and their assigned "keys". It is used when client authentication is required.

The JBoss Enterprise SOA Platform ships with an example key-store, found in **`SOA_ROOT/jboss-as/samples/quickstarts/security_cert/keystore`**. Do not use this in a production environment. It is provided as an example only.

Report a bug

7.16. JBOSS RULES AND SECURITY

By default, the JBoss Rules component does not deserialize rules packages or unsigned rule bases.

IMPORTANT

You must activate this serialization security feature in order for your configuration to be supported by Red Hat. You need to configure system properties for both the application that serializes the packages and its rule bases (hereafter referred to as the server), as well as the application that deserializes the packages its rule bases (hereafter referred to as the client).

Report a bug

7.17. ENABLE SERIALIZATION ON THE SERVER

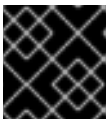
Procedure 7.6. Task

1. Navigate to the SOA_ROOT directory: **`cd SOA_ROOT`**.
2. Run the **`keytool`** command and follow the prompts on screen:

```
keytool -genkey -alias droolsKey -keyalg RSA -keystore
MyDroolsPrivateKeyStore.keystore
Enter keystore password:
Re-enter new password:
What is your first and last name?
  [Unknown]:  Test User
What is the name of your organizational unit?
  [Unknown]:  HR
What is the name of your organization?
  [Unknown]:  Test Org
What is the name of your City or Locality?
  [Unknown]:  Brisbane
```

```
What is the name of your State or Province?
  [Unknown]:  QLD
What is the two-letter country code for this unit?
  [Unknown]:  AU
Is CN=Test User, OU=HR, O=Test Org, L=Brisbane, ST=QLD, C=AU
correct?
  [no]:  yes
Enter key password for droolsKey
     (RETURN if same as keystore password):
Re-enter new password:
```

After answering all of the questions, a password-protected file named **MyDroolsPrivateKeyStore.keystore** is created. This keystore file has a private key called droolsKey with the password "drools". Store this file in a safe location in your environment, which will hereafter be referred to as the **keystoredir**.



IMPORTANT

The passwords above are examples only and should not be used in production.

3. Open the configuration file: **vi jboss-as/server/default/deploy/properties-service.xml**

4. Configure the JBoss Enterprise SOA Platform to use the JBoss Rules serialization feature by adding this snippet to **properties-service.xml**:

```
<mbean code="org.jboss.varia.property.SystemPropertiesService"
name="jboss:type=Service,name=SystemProperties">
    <attribute name="Properties">
      # Drools Security Serialization specific properties
      drools.serialization.sign=true

drools.serialization.private.keyStoreURL=file://$keystoredir/MyDrool
sPrivateKeyStore.keystore
      drools.serialization.private.keyStorePwd=drools
      drools.serialization.private.keyAlias=droolsKey
      drools.serialization.private.keyPwd=drools
    </attribute>
</mbean>
```

5. Set the drools.serialization.sign property to "true":

```
drools.serialization.sign=true
```

   - drools.serialization.private.keyStoreURL=<RL> is the URL of the private keystore location.
   - In the example above, replace **keystoredir** and **MyDroolsKeyStore.keystore** with your keystore directory and the name of the keystore you created with the keytool
   - drools.serialization.private.keyStorePwd=<password> is the password to access the private keystore.
   - drools.serialization.private.keyAlias=<key> is the key alias (identifier) of the private key.

- drools.serialization.private.keyPwd=<password> is the private key password.

6. Save the file and exit.
7. Restart the server instance.

WARNING

If the system properties were not configured properly, you will see this error when you try to build a rules package:

```
An error occurred building the package.

Error
signing object store: Key store with private key not
configured. Please
configure it properly before using signed
serialization
```

Report a bug

7.18. ENABLE SERIALIZATION ON THE CLIENT

Prerequisites

- Server serialization must already be enabled.

Procedure 7.7. Task

1. Create a public key certificate from the private keystore. (You can access the keytool by running **`keytool -genkey -alias droolsKey -keyalg RSA -keystore`**.):

```
keytool -export -alias droolsKey -file droolsKey.crt -keystore
```

```
MyDroolsPrivateKeyStore.keystore
Enter keystore password:
Certificate stored in file <droolsKey.crtU>
```

2. Import the public key certificate into a public keystore. (This is where it will be used by your client applications):

```
keytool -import -alias droolsKey -file droolsKey.crt -keystore
```

```
MyPublicDroolsKeyStore.keystore
Enter keystore password:
```

```
Re-enter new password:
Owner: CN=Test User, OU=Dev, O=XYZ Corporation, L=Brisbane, ST=QLD,
C=AU
Issuer: CN=Test User, OU=Dev, O=XYZ Corporation, L=Brisbane, ST=QLD,
C=AU
Serial number: 4ca0021b
Valid from: Sun Sep 26 22:31:55 EDT 2010 until: Sat Dec 25 21:31:55
EST 2010
Certificate fingerprints:
     MD5:  31:1D:1B:98:59:CC:0E:3C:3F:57:01:C2:FE:F2:6D:C9
     SHA1:
4C:26:52:CA:0A:92:CC:7A:86:04:50:53:80:94:2A:4F:82:6F:53:AD
     Signature algorithm name: SHA1withRSA
     Version: 3
Trust this certificate? [no]:  yes
Certificate was added to keystore
```

3. Open the server configuration file: **`vi grep drools jboss-as/server/default/deploy/properties-service.xml`**

4. Replace keystoredir and MyPublicDroolsKeyStore.keystore with your keystore directory, and the name of the public keystore you created previously:

```
# Drools Client Properties for Security Serialization
drools.serialization.public.keyStoreURL=file://$keystoredir/MyPublic
DroolsKeyStore.keystore
drools.serialization.public.keyStorePwd=drools
```

5. Save the file and exit.

6. Restart the JBoss Enterprise SOA Platform server.

7. For Java client applications, set the system properties in your code like this:

```
// Set the client properties to deserialize the signed packages
URL clientKeyStoreURL = getClass().getResource(
"MyPublicDroolsKeyStore.keystore" );
System.setProperty( KeyStoreHelper.PROP_SIGN,
                            "true" );
System.setProperty( KeyStoreHelper.PROP_PUB_KS_URL,
                            clientKeyStoreURL.toExternalForm() );
System.setProperty( KeyStoreHelper.PROP_PUB_KS_PWD,
                            "drools" );
...
```

Alternatively, open the **`run.sh`** shell script (**`vi SOA_ROOT/jboss-as/bin/run.sh`**) script and edit the JAVA_OPTS section:

```
# Serialization Security Settings
JAVA_OPTS="-Ddrools.serialization.sign=true $JAVA_OPTS"
JAVA_OPTS="-
Ddrools.serialization.private.keyStoreURL=file://$keystoredir/MyDroo
lsKeyStore.keystore $JAVA_OPTS"
JAVA_OPTS="-Ddrools.serialization.private.keyStorePwd=drools
$JAVA_OPTS"
```

```
JAVA_OPTS="-Ddrools.serialization.private.keyAlias=droolsKey
$JAVA_OPTS"
JAVA_OPTS="-Ddrools.serialization.private.keyPwd=drools $JAVA_OPTS"
JAVA_OPTS="-
Ddrools.serialization.public.keyStoreURL=file://$keystoredir/MyPubli
cDroolsKeyStore.keystore $JAVA_OPTS"
JAVA_OPTS="-Ddrools.serialization.public.keyStorePwd=drools
$JAVA_OPTS"
```

Replace the values shown above with ones specific to your environment, and then restart the server instance.

Report a bug

7.19. DISABLE SERIALIZATION SIGNING

1. Open the configuration file: **`vi SOA_ROOT/jboss-as/server/PROFILE/deploy/properties-service.xml`**.
2. Remove the drools.serialization.sign property's value.
3. Save the file and exit.

   An alternative way to do this task is to open the **`run.sh`** shell script (**`vi SOA_ROOT/jboss-as/bin/run.sh`**) and edit it as follows:

   ```
   JAVA_OPTS="-Ddrools.serialization.sign=false $JAVA_OPTS"
   ```

4. Restart the server instance.
5. To turn signing off for Java client applications, remove the drools.serialization.sign property or add the following snippet to each application's code:

   ```
   System.setProperty( KeyStoreHelper.PROP_SIGN, "false" );
   ```

Report a bug

7.20. CONFIGURE SECURITY ON A PER-SERVICE BASIS

1. Open the global configuration file in a text editor: **`vi SOA_ROOT/jboss-as/server/PROFILE/deployers/esb.deployer/jboss-esb.xml`**.
2. Scroll down to the service you want to configure.
3. Add a security element. This setting shows you how to do so:

   ```
   <service category="Security" name="SimpleListenerSecured">
      <security moduleName="messaging" runAs="adminRole"
       rolesAllowed="adminRole, normalUsers"

   callbackHandler="org.jboss.internal.soa.esb.services.security.UserPa
   ```

```
ssCallbackHandler">
     <property name="property1" value="value1"/>
     <property name="property2" value="value2"/>
   </security>
   ...
</service>
```

4. Save the file and exit.

Report a bug

7.21. PER-SERVICE SECURITY PROPERTIES

Table 7.3. Security Properties

Property	Description	Required?
moduleName	This is a module that exists in the **SOA_ROOT/jboss-as/server/PROFILE/conf/login-config.xml** file.	No
runAs	This is the runAs role.	No
rolesAllowed	This is an comma-separated list of those roles that have been granted the ability to execute the service. This is used as a check that is performed after a caller has been authenticated, in order to verify that they are indeed belonging to one of the roles specified. The roles will have been assigned after a successful authentication by the underlying security mechanism.	No
callbackHandler	This is the **CallbackHandler** that will override that which was defined in the **jbossesb-properties.xml** file.	No
property	These are optional properties that, once defined, will be made available to the **CallbackHandler** implementation.	No

Report a bug

7.22. OVERRIDE GLOBAL SECURITY SETTINGS

Procedure 7.8. Task

1. Open the global configuration file in a text editor: **`vi SOA_ROOT/jboss-as/server/PROFILE/deployers/esb.deployer/jbossesb-properties.xml`**.

2. Configure the setting in question. Here is an example:

```
<security moduleName="messaging"
  runAs="adminRole" rolesAllowed="adminRole">

<property
    name="org.jboss.soa.esb.services.security.contextTimeout"
    value="50000"/>

<property name=
"org.jboss.soa.esb.services.security.contextPropagatorImplementation
Class"
    value="org.xyz.CustomSecurityContextPropagator" />

</security>
```

3. Save the file and exit.

Report a bug

7.23. SECURITY PROPERTY OVERRIDES

Table 7.4. Security Property Overrides:

Property	Description	Required?
org.jboss.soa.esb.services.security.contextTimeout	This property lets the service override the global security context time-out (milliseconds) that is specified in the **`jbossesb-properties.xml`** file.	No
org.jboss.soa.esb.services.security.contextPropagatorImplementationClass	This property lets the service to override the "global security context propagator" class implementation, that is specified in the **`jbossesb-properties.xml`** file.	No

Report a bug

7.24. SECURITY CONTEXT

The SecurityContext is an object which is created after a security certificate is confirmed. After creation, it will be configured so that you do not have to re-authenticate the certificate every time you perform an action related to it. If the ESB detects that a message has a SecurityContext, it will check that it is still

valid and, if so, it does not try to re-authenticate it. (Note that the SecurityContext is only valid for a single Enterprise Service Bus node. If the message is routed to a different ESB node, it will have to be re-authenticated.)

Report a bug

7.25. AUTHENTICATION REQUEST

An AuthenticationRequest carries the security information needed for authentication between either a gateway and a service or between two services. You must set an instance of this class on the message object prior to the authenticating service being called. The class must contain the principle and the credentials needed to authenticate a caller. This class is made available to the Callback Handler.

Report a bug

7.26. SECURITYCONFIG

The **`SecurityConfig`** class grants access to the security configuration specified in the **`jboss-esb.xml`** file. This class is made available to the Callback Handler.

Report a bug

7.27. ADD AN AUTHENTICATION CLASS TO A MESSAGE OBJECT

Procedure 7.9. Task

- Execute this code:

```
byte[] encrypted = PublicCryptoUtil.INSTANCE.encrypt((Serializable)
 authRequest);
message.getContext.setContext(SecurityService.AUTH_REQUEST,
encrypted);
```

Result

The authentication context is encrypted and then set within the message context. (It is later decrypted by the Enterprise Service Bus so that it can authenticate the request.)

Report a bug

7.28. SECURITY_BASIC QUICK START

The **`SOA_ROOT/jboss-as/samples/quickstarts/security_basic`** quick start demonstrates how to prepare the security on a message before you use the SecurityInvoker. The quick start also demonstrates how to configure the **`jbossesb-properties.xml`** global configuration file for use by client services.

Report a bug

7.29. SET A TIME LIMIT FOR THE SECURITY CONTEXT GLOBALLY

Procedure 7.10. Task

1. Open the global configuration file in a text editor: **`vi SOA_ROOT/jboss-as/server/PROFILE/deployers/esb.deployer/jbossesb-properties.xml`**.
2. Scroll down to the section that contains security.contextTimeout. Set the time-out value (in milliseconds).
3. Save the file and exit.

Report a bug

7.30. SET A TIME LIMIT FOR THE SECURITY CONTEXT ON A PER-SERVICE BASIS

Procedure 7.11. Task

1. Open the service's configuration file in a text editor: **`vi jboss-esb.xml`**.
2. Scroll down to the section that contains Security Context. Set the time-out value (in milliseconds).
3. Save the file and exit.

Report a bug

CHAPTER 8. ADVANCED SECURITY OPTIONS

8.1. SECURITY PROPAGATION

The term "security propagation" refers to the process of passing security information to an external system. For example, you might want to use the same credentials to call both the Enterprise Service Bus and an Enterprise Java Beans method.

Report a bug

8.2. SECURITYCONTEXTPROPAGATOR

The SecurityContextPropagator class passes the security context to the destination environment.

Report a bug

8.3. SECURITYCONTEXTPROPAGATOR IMPLEMENTATIONS

Table 8.1. Implementations of SecurityContextPropagator

Class	Description
Package: org.jboss.internal.soa.esb.services.security Class: JBossASContextPropagator	This propagator will send security credentials to the ESB. If you need to write your own implementation you only have to write a class that implements **`org.jboss.internal.soa.esb.services.security.SecurityContextPropagator`** and then either specify that implementation in **`jbossesb-properties.xml`** or **`jboss-esb.xml`**.

Report a bug

8.4. ADD A CUSTOM LOG-IN MODULE

Procedure 8.1. Task

1. Open the log-in configuration file in a text editor: **`vi SOA_ROOT/jboss-as/server/PROFILE/conf/login-config.xml`**
2. Add the details of your custom log-in module.
3. Save the file and exit.
4. Since different log-in modules require different information, you must specify the CallbackHandler attribute to be used. Open the specific security configuration for that service.
5. Make sure that the **`CallbackHandler`** specifies a *fully-qualified classname* for the class which implements the **`EsbCallbackHandler`** interface. This code shows you how to do so:

```
public interface EsbCallbackHandler extends CallbackHandler
{
  void setAuthenticationRequest(final AuthenticationRequest
authRequest);
  void setSecurityConfig(final SecurityConfig config);
}
```

6. Add both the "principle" and the credentials needed to authenticate a caller to the **`AuthenticationRequest`** class.

Result

JaasSecurityService is replaced with your custom security implementation.

Report a bug

8.5. CERTIFICATE LOG-IN MODULE

The Certificate Log-in Module performs authentication by verifying the certificate that is passed with the call to the Enterprise Service Bus against a certificate held in a local key-store. The certificate's common name creates a "principle".

Report a bug

8.6. CERTIFICATE LOG-IN MODULE PROPERTIES

```
<security moduleName="CertLogin" rolesAllowed="worker"
  callbackHandler="org.jboss.soa.esb.services.security.auth.loginUserPass
CallbackHandler">
  <property name="alias" value="certtest"/>
</security>
```

Table 8.2. Properties

Property	Description
moduleName	This identifies the JAAS Login module to use. This module will be specified in JBossAS login-config.xml.
rolesAllow	This is a comma-separated list of the roles that are allowed to execute this service.
alias	This is the alias which is used to look up the local key-store and which will be used to verify the caller's certificate.

Report a bug

8.7. CERTIFICATE LOG-IN MODULE CONFIGURATION FILE PROPERTIES

```
<application-policy name="CertLogin">
<authentication>
  <login-module
code="org.jboss.soa.esb.services.security.auth.login.CertificateLoginModul
e"
flag = "required" >
  <module-option name="keyStoreURL">
    file://pathToKeyStore
  </module-option>
  <module-option name="keyStorePassword">storepassword</module-option>
  <module-option name="rolesPropertiesFile">
    file://pathToRolesFile
  </module-option>
  </login-module>
</authentication>
</application-policy>
```

Table 8.3. Certificate Log-In Module Configuration File Properties

Property	Description
keyStoreURL	This is the path to the key-store used to verify the certificates. It can be a file on the local file system or on the class-path.
keyStorePassword	This is the password for the key-store above.
rolesPropertiesFile	This is optional. It is the path to a file containing role mappings. Refer to the “Role Mapping” section of the Getting Started Guide for more details about this.

Report a bug

8.8. CALLBACK HANDLER

A callback handler is a type of library used in back-end operations. It allows applications to "talk" to each other through security services and can be used to confirm authentication data.

Report a bug

8.9. ROLE MAPPING

Role mapping is a way of sharing data between secure hosts. A file containing a list of trusted hosts is created, with each host assigned several role mappings. Mapping occurs when you accesses data from one of the hosts. The sender's roles are mapped onto the receiver's roles to allow for authentication and data sharing. Types of roles include user roles, application roles and so forth. This is an optional feature and is not enabled by default.

Report a bug

8.10. ENABLE ROLE MAPPING

Procedure 8.2. Task

1. Open the log-in configuration file in a text editor: **`vi SOA_ROOT/jboss-as/server/PROFILE/conf/login-config.xml`**
2. Set the rolesPropertiesFile property. (This property can point to a file located on either the local file system or the class-path).
3. Map users to roles. This example code shows how to do so:

```
# user=role1,role2,...
guest=guest
esbuser=esbrole
# The current implementation will use the Common Name(CN) specified
# for the certificate as the user name.
# The unicode escape is needed only if your CN contains a space
Andy\u0020Anderson=esbrole,worker
```

4. Save the file and exit.

Report a bug

8.11. SECURITY_CERT QUICKSTART

The security_cert quickstart demonstrates the JBoss Enterprise SOA Platform's role-mapping functionality.

Report a bug

8.12. SECURITY SERVICE

The **`SecurityService`** interface is the Enterprise Service Bus' central security component.

Report a bug

8.13. CUSTOMIZE THE SECURITY SERVICE INTERFACE

Procedure 8.3. Task

1. Implement the **`SecurityService`** interface:

```
public interface SecurityService
{
    void configure() throws ConfigurationException;
```

```
    void authenticate(
        final SecurityConfig securityConfig,
        final SecurityContext securityContext,
        final AuthenticationRequest authRequest)
        throws SecurityServiceException;

    boolean checkRolesAllowed(
        final List<String> rolesAllowed,
        final SecurityContext securityContext);

    boolean isCallerInRole(
        final Subject subject,
        final Principle role);

    void logout(final SecurityConfig securityConfig);

    void refreshSecurityConfig();
}
```

2. Open the global configuration file in a text editor: **`vi SOA_ROOT/jboss-as/server/PROFILE/deployers/esb.deployer/jbossesb-properties.xml`**.

3. Configure the file to use the customized **`SecurityService`**

4. Save the file and exit.

Report a bug

8.14. REMOTE INVOCATION CLASS

As its name implies, a remote invocation class is a class that can be called from a remote machine. This can be useful for developers but can also lead to potential security risks.

Report a bug

8.15. SECURE NON-REMOTE METHOD INVOCATION CLASSES ON PORT 8083

Client applications can, by default, utilize Remote Method Invocation to download Enterprise Java Bean classes through **`port 8083`**. However, you can also configure the system's Remote Method Invocation settings to allow client applications to download any deployed resources you desire.

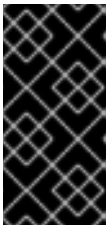
Procedure 8.4. Task

1. **Edit the Settings in the jboss-service.xml File**
   Open the file in a text editor: **`vi SOA_ROOT/server/PROFILE/conf/jboss-service.xml`**

2. **Configure the Settings in the File**
   Here is an example:

```
<attribute name="DownloadServerClasses">false</attribute>
```

Set this value to false to ensure that client applications can only download Enterprise Java Bean classes.



IMPORTANT

By default, this value is set to false in the SOA Platform's 'production' profile. The value is set to true in all other cases, including the SOA Standalone version's default profile. Note that this is not a secure configuration and should only be used in development environments.

Report a bug

CHAPTER 9. SECURING THE SERVICE REGISTRY

9.1. JUDDI AND THE JBOSS ENTERPRISE SOA PLATFORM

jUDDI and the JBoss Enterprise SOA Platform

The JBoss Enterprise SOA Platform product includes a pre-configured installation of a jUDDI registry. You can use a specific API to access this registry through your custom client. However, any custom client that you build will not covered by your SOA Platform support agreement. You can access the full set of jUDDI examples, documentation and APIs from: http://juddi.apache.org/.

Report a bug

9.2. SERVICE REGISTRY AUTHENTICATION

Introduction

Here is a theoretical understanding of how the authentication process works.

Authentication is a two-phase process. These are known as the *authenticate phase* and the *identify phase*. Both of these phases are represented by a method in the **`Authenticator`** interface.

The authenticate phase occurs when the **`GetAuthToken`** request is made. The goal of this phase is to turn a user id and credentials into a valid publisher id. The publisher id (referred to as the *authorized name* in UDDI terminology) is the value that assigns ownership within UDDI. Whenever a new entity is created, it must be tagged with ownership by the authorized name of the publisher.

The value of the publisher id is irrelevant to the jUDDI Registry: the only requirement is that one exists to assign to new entities so it must be non-null. Upon completion of the **`GetAuthToken`** request, an **`authentication token`** is issued to the caller.

When you make subsequent calls to the UDDI API that require authentication, you must provide the token issued in response to the **`GetAuthToken`** request. This leads to the identify phase.

The identify phase is responsible for turning the authentication token (or the publisher id associated with that token) into a valid **`UddiEntityPublisher`** object. This object contains all the properties necessary to handle ownership of UDDI entities. Thus, the token (or publisher id) is used to identify the publisher.

The two phases provide compliance with the UDDI authentication structure and grant flexibility if you wish to provide your own authentication mechanism.

Handling of credentials and publisher properties could be done entirely outside of the jUDDI Registry. However, by default, the Registry provides the **`Publisher`** entity, which is a sub-class of **`UddiEntityPublisher`**. This sub-class makes publisher properties persist within the jUDDI Registry.

Report a bug

9.3. AUTHTOKEN

An authToken is a security container holding password credentials.

Report a bug

9.4. AUTHTOKEN AND THE SERVICE REGISTRY

In order to enforce proper write access to the Service Registry, each request made to it needs a valid **`authToken`**.

IMPORTANT

Note that read access is not restricted at all.

Report a bug

9.5. OBTAIN AN AUTHTOKEN

Procedure 9.1. Task

1. Make a **`GetAuthToken()`** request.
2. A **`GetAuthToken`** object is returned. Set a userid and credential (password) on this object:

```
org.uddi.api_v3.GetAuthToken ga = new
org.uddi.api_v3.GetAuthToken();
ga.setUserID(pubId);
ga.setCred("");

org.uddi.api_v3.AuthToken token = securityService.getAuthToken(ga);
```

3. Locate the **`juddi.properties`** configuration file in **`SOA_ROOT/jboss-as/server/PROFILE/deploy/juddi-service.sar/juddi.war/WEB-INF`**. Open it in a text editor.
4. Configure the juddi.authenticator property to how the Service Registry will check the credentials passed to it by the **`GetAuthToken`** request. (By default it uses the **`jUDDIAuthenticator`** implementation.)
5. Save the file and exit.

Report a bug

9.6. SECURITY AUTHENTICATION IMPLEMENTATIONS AVAILABLE FOR THE SERVICE REGISTRY

jUDDI Authentication

WARNING

Do not use this authentication method in a production environment. It accepts any credentials provided, and effectively removes the need for clients to authenticate when accessing the registry.

The default authentication mechanism provided by the Service Registry is the **`jUDDIAuthenticator`**. **`jUDDIAuthenticator`**'s authenticate phase checks to see if the, user ID submitted matches against a record in the **`Publisher`** table. No credentials checks are made. If, during the authentication process, the Publisher record is found to be non-existent, it is added "on-the-fly".

In the identify phase, the publisher ID is used to retrieve the Publisher record and return it. The Publisher inherits every property it needs from **`UddiEntityPublisher`**:

```
juddi.authenticator = org.apache.juddi.auth.JUDDIAuthentication
```

XMLDocAuthentication

The authenticate phase checks that the user id and password match a value in the XML file. The identify phase uses the user ID to populate a new **`UddiEntityPublisher`**.

CryptedXMLDocAuthentication

The **`CryptedXMLDocAuthentication`** implementation is similar to the **`XMLDocAuthentication`** implementation, but the passwords are encrypted:

```
juddi.authenticator = org.apache.juddi.auth.CryptedXMLDocAuthentication
juddi.usersfile = juddi-users-encrypted.xml
juddi.cryptor = org.apache.juddi.cryptor.DefaultCryptor
```

Here, the user credential file is **`juddi-users-encrypted.xml`**, and the content of the file will be similar to this:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<juddi-users>
<user userid="anou_mana" password="+j/kXkZJftwTFTBH6Cf6IQ=="/>
<user userid="bozo" password="Na2Ait+2aW0="/>
<user userid="sviens" password="+j/kXkZJftwTFTBH6Cf6IQ=="/>
</juddi-users>
```

The **`DefaultCryptor`** implementation uses **`BEWithMD5AndDES`** and **`Base64`** to encrypt the passwords.

NOTE

You can use the code in the **`AuthenticatorTest`** to learn more about how to use this Authenticator implementation. You can plug in your own encryption algorithm by implementing the **`org.apache.juddi.cryptor.Cryptor`** interface and referencing your implementation class in the juddi.cryptor property.

The authenticate phase checks that the user ID and password match values in the XML file. The identify phase uses the user ID to populate a new **`UddiEntityPublisher`**.

LDAP Authentication

Use **`LdapSimpleAuthenticator`** to authenticate users via LDAP's simple authentication functionality. This class allows you to authenticate a user based on an *LDAP principle*, provided that the principle and the jUDDI publisher ID are identical.

JBoss Authentication

A final alternative is to interface with third-party credential stores. You can link it to the JBoss Application Server's authentication component.

You will find the **`JBossAuthenticator`** class provided in the **`docs/examples/auth`** directory. This class enables jUDDI deployments on JBoss to use a server security domain to authenticate users.

Report a bug

9.7. CONFIGURE XMLDOCAUTHENTICATION

Procedure 9.2. Task

1. Create a text file called **`juddi-users.xml`** and save it in **`jbossesb-registry.sar`**.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<juddi-users>
    <user userid="sscholl" password="password" />
    <user userid="dsheppard" password="password" />
    <user userid="vbrittain" password="password" />
</juddi-users>
```

2. Save the file and exit.
3. Add the file to the class-path.
4. Open the **`juddi.properties`** file in your text editor (located in **`SOA_ROOT/jboss-as/server/PROFILE/deploy/juddi-service.sar/juddi.war/WEB-INF`**).
5. Modify the file so that it looks like this:

```
juddi.authenticator = org.apache.juddi.auth.XMLDocAuthentication
juddi.usersfile = juddi-users.xml
```

6. Save the file and exit.

Report a bug

9.8. LIGHTWEIGHT DIRECTORY ACCESS PROTOCOL (LDAP)

Lightweight Directory Access Protocol (LDAP) is a protocol for accessing distributed directory information over the internet.

Report a bug

9.9. CONFIGURE LDAP AUTHENTICATION

Procedure 9.3. Task

1. Locate the **juddi.properties** file in **SOA_ROOT/jboss-as/server/PROFILE/deploy/juddi-service.sar/juddi.war/WEB-INF**. Open it in your text editor.
2. Add the following configuration settings:

```
juddi.authenticator=org.apache.juddi.auth.LdapSimpleAuthenticator
juddi.authenticator.url=ldap://localhost:389
```

   The juddi.authenticator.url property tells the **LdapSimpleAuthenticator** class where the LDAP server resides.
3. Save the file and exit.

Report a bug

9.10. CONFIGURE JBOSS AUTHENTICATION

Procedure 9.4. Task

1. Locate the **juddi.properties** file in **SOA_ROOT/jboss-as/server/PROFILE/deploy/juddi-service.sar/juddi.war/WEB-INF**. Open it in your text editor.
2. Add the following lines to the file:

```
uddi.auth=org.apache.juddi.auth.JBossAuthenticator
juddi.securityDomain=java:/jaas/other
```

   The juddi.authenticator property connects the **JbossAuthenticator** class to the jUDDI Registry's Authenticator framework. The **juddi.security.domain** tells **JBossAuthenticator** where it can find the Application Server's security domain. It uses this domain to perform the authentications.

Note that JBoss creates one security domain for each application policy element in the **SOA_ROOT/jboss-as/server/PROFILE/conf/login-config.xml** file. These domains are bound to the server JNDI tree with this name: **java:/jaas/<application-policy-name>**. (If a look-up refers to a non-existent application policy, a policy named **other** will be used by default.)

3. Save the file and exit.

Report a bug

PART III. WEB CONSOLES

CHAPTER 10. MONITORING YOUR SYSTEM WITH THE ADMIN WEB CONSOLE

10.1. ADMIN CONSOLE

The JBoss SOA Platform (SOA-P) Admin Console is a web-based tool allowing system administrators to monitor and configure services deployed within a running SOA-P instance.

Report a bug

10.2. RUNNING THE ADMIN CONSOLE

Prerequisites

- JBoss Enterprise SOA Platform must be installed and running.
- Your user details must be correctly configured in:
  - ***SOA_ROOT*/jboss-as/server/*PROFILE*/conf/props/soa-users.properties** and
  - ***SOA_ROOT*/jboss-as/server/*PROFILE*/conf/props/soa-roles.properties**.

Procedure 10.1. Running the Admin Console

1. **Launch the Console in a Web Browser**
   Open http://localhost:8080/admin-console in a web browser.
2. **Authenticate to the Console**
   Enter your **Username** and **Password** as set in ***SOA_ROOT*/jboss-as/server/*PROFILE*/conf/props/soa-users.properties**.

Report a bug

10.3. VIEW A QUEUE IN THE ADMIN CONSOLE

Procedure 10.2. Task

1. Launch a web browser and go to localhost:8080/admin-console.
2. Input **admin** as the username and password.
3. To view the queue, click on **Resources**, **JBoss Messaging** and then **Queues**.

Result

A list of the JMS queues deployed on the server appears.

Report a bug

CHAPTER 11. MONITORING YOUR SYSTEM WITH THE SERVICE LIST CONSOLE

11.1. SERVICE LIST CONSOLE

The Service List Console allows system administrators to view information about the services listed in the JBoss Enterprise SOA Platform's service registry. Access it from http://localhost:8080/contract/.

Report a bug

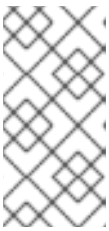
11.2. SERVICE LIST CONSOLE FUNCTIONALITY

The Service List Console displays the following information:

- processing times
- numbers of failed messages
- bytes transferred
- the date-time stamps of the last successful and failed messages

It also displays the following information for each ESB service:

- processing time per action
- processed count per action
- failed count per action
- overall message count (per service)
- The Console also keeps count of the number of messages that have passed through the Enterprise Service Bus via a message counter. (This counter also tracks the numbers of successfully-processed and failed messages and records the number of bytes processed and the time-stamp for each message.)
- You can also monitor Dead Letter Service which handles undeliverable messages.



NOTE

The Dead Letter Service will not, however, be used if the underlying transport has native support. (This is the case for the Java Messaging Service.) In these situations, you must inspect both the Dead Letter Service and any transport-specific equivalent.

- The Console also keeps track of the events performed on the Action Pipeline, including Smooks transformations (and the amount of time taken to perform them).

Report a bug

CHAPTER 12. MONITORING YOUR SYSTEM WITH THE JMX CONSOLE

12.1. JMX CONSOLE

The JMX Console is a web console for monitoring Java message transactions. You can deploy various M-Beans that gather a miscellany of performance statistics. Access the JMX Console at http://localhost:8080/jmx-console. It allows you to view inside the application server's microkernel to see all of the registered services (which take the form of M-Beans).

Report a bug

12.2. M-BEAN

An M-Bean (Managed Bean) is a Java object that represents a manageable resource, such as a service or application. All of the registered services in the application server's micro-kernel are represented as M-Beans.

Report a bug

12.3. MONITORING AND MANAGEMENT M-BEANS

In the jboss.esb> domain, you will find the following M-Beans:

deployment=<ESB package name>

Use the Deployments M-Bean to see the status of every deployed ESB package and its associated XML configuration.

listener-name=<Listener name>

This M-Bean lists all of the deployed listeners. It shows information about their XML configurations, their start times, their maxThreads and their states.

If your listener has an explicitly-managed thread pool , its current minimum and maximum thread pool counts will also be exposed through this MBean.

The number of active threads in the thread pool will change dynamically between this minimum (which is initially set to one) and the defined maximum as the service load dictates. The administrator has the option of changing these values while the system is running although they will revert to their original values should the server, or ESB artifact, be restarted.

From here, you can also initialize, start, stop and destroy them.

category=MessageCounter

The message counters display all of the services deployed for a given listener, the actions for each of these services, the number of messages processed and the time taken to process each one.

service-name=<Service name>

This M-Bean displays a variety of statistics for each service, including message counts, state, average size and processing time. You can reset message counts and start and stop services from here as well.

NOTE

In addition to the M-Beans listed above, the Java Message Service domain provides some extra ones. These show statistics for message queues. You might find this information useful when you are debugging the system or analysing its performance.

Report a bug

CHAPTER 13. MONITORING YOUR SYSTEM WITH THE JON FOR SOA WEB CONSOLE

13.1. JBOSS OPERATIONS NETWORK (JON)

The JBoss Operations Network (JON) program is a tool that you can use to administer your JBoss Enterprise SOA Platform server.

It enables you to perform inventorying, monitoring, deployment and software update administrative tasks. JON utilizes a customizable web-portal interface and it provides a central place where you can manage your system.

Report a bug

13.2. JON FOR SOA

JON for SOA is a specially-packaged version of JON that includes functionality designed specifically for the JBoss Enterprise SOA Platform.

Report a bug

13.3. ANALYSE JBOSS ENTERPRISE SOA PLATFORM ENTERPRISE SERVICE BUS STATISTICS

Procedure 13.1. Task

1. Click on **JBoss ESB Statistics** (above the **`Resources`** menu) to "drill down" through various levels of statistics.

   On the first level, the figures displayed are a summary for the overall ESB instance.

2. Click on the **`JBoss ESB Deployment`** item to view a list of all of the Enterprise Service Bus packages deployed on the server. One will not see any statistics at this level but, from here, one can select a deployment and drill down into it to view them.

3. Drill down further still to view details for that deployment's constituent services and actions.

Report a bug

13.4. METRICS AVAILABLE THROUGH JON FOR SOA

Different metrics are displayed depending on the level selected. The following lists summarises everyone that is available:

Statistics Available at the ESB Level:

- Message Count (Successful)

- Message Count (Total)
- Message Counts (Failed)
- Processed Bytes
- Last Failed Message Date
- Last Successful Message Date

Statistics Available at the Service Level:

- Message Count
- Message Count (avg) per Minute
- Overall Bytes
- Overall Bytes Failed
- Overall Bytes Processed
- Overall Service Time Processed

Statistics Available at the Action Level:

- Message Count
- Message Count (avg) per Minute
- Messages Failed
- Messages Failed (avg) per Minute
- Messages Successfully Processed
- Messages Successfully Processed (avg) per Minute
- Overall Bytes
- Overall Bytes Failed
- Overall Bytes Processed
- Processing Time

Statistics Available at the Listener Level:

- Life-cycle State
- Maximum number of threads
- MEP
- Service Category
- Service Description

- Service Name
- Start Date

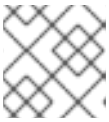
Use these statistics to configure any standard JBoss Operations Network function, (such as alerts), for any JBoss Enterprise SOA Platform deployment, service or action.

Report a bug

13.5. USE JON FOR SOA TO DEPLOY AN ARCHIVE

Procedure 13.2. Task

1. Open a web browser and log into the JON for SOA Console.
2. Go to the **`JBoss ESB Statistics`** screen.
3. Click on the **`INVENTORY`** tab
4. Go to **`Child Resources`**.



NOTE

You can view historical deploy requests here as well.

5. Go to the **`Create New`** menu and select **JBoss ESB Deployment**.
6. On the **`Create New Resource`** page, choose the archive to deploy and select where it should be sent (which, under normal circumstance, will be your **`deploy`** directory).

NOTE

Remember that only compressed files can be uploaded: use the **`Deploy Zipped`** option to determine whether it should be deployed as a compressed or an exploded archive.

Report a bug

13.6. USE JON FOR SOA TO DELETE AN ARCHIVE

Procedure 13.3. Task

1. Open a web browser and log into the JON for SOA Console.
2. Go to the **`JBoss ESB Statistics`** screen.
3. Click on the **`INVENTORY`** tab
4. Go to **`Child Resources`** list.

5. Tick the entry to be deleted.
6. Click **DELETE**

NOTE

You can view historical delete requests here as well.

Result

The archive is deleted.

Report a bug

13.7. AUTOMATIC SERVICE DISCOVERY

The JON agent can automatically detect ESB archives deployed or deleted independently of JON. This is known as the Automatic Service Discovery feature. The JON agent performs this check once every twenty-four hours.

Report a bug

13.8. CHANGE THE AUTOMATIC SERVICE DISCOVERY FEATURE'S POLLING RATE

Procedure 13.4. Task

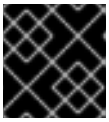
1. Open the configuration file installed with the JON Agent in a text editor: **`vi rhq-agent/conf/agent-configuration.xml`**

   Edit the file as per this example code:

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

2. Save the file and exit.
3. Restart the JBoss Enterprise SOA Platform.

   In contrast to the JBoss Enterprise SOA Platform consoles, there is no way to force the JON web console to collect new data on demand. Clicking on buttons such as **`Get Current Values`** (found under the **`Metric Data`** tab) only updates the display to reflect the most recently collected data. If you want an an update immediately, reset the collection period to a very low value, such as thirty seconds. (Remember to set the interval back to the previous figure afterwards.)



IMPORTANT

If you set the value to too low a figure, performance will suffer.

Report a bug

13.9. CHANGE THE AUTOMATIC SERVICE DISCOVERY FEATURE'S POLLING RATE (ALTERNATIVE METHOD)

Procedure 13.5. Task

1. Open a web browser and log into the JON for SOA console.
2. Add a JON agent to the server's inventory of resources.
3. Click on **CONFIGURE**.
4. Change the value for **Service Discovery Period**.

NOTE

You do not need to restart the agent for the change to take effect.

Report a bug

CHAPTER 14. ADMINISTERING YOUR SERVICE REGISTRY WITH THE JUDDI WEB CONSOLE

14.1. SERVICE REGISTRY

A service registry is a central database that stores information about services, notably their end-point references. The default service registry for the JBoss Enterprise SOA Platform is jUDDI (Java Universal Description, Discovery and Integration). Most service registries are designed to adhere to the Universal Description, Discovery and Integration (UDDI) specifications.

From a business analyst's perspective, the registry is similar to an Internet search engine, albeit one designed to find web services instead of web pages. From a developer's perspective, the registry is used to discover and publish services that match various criteria.

In many ways, the Registry Service can be considered to be the "heart" of the JBoss Enterprise SOA Platform. Services can "self-publish" their end-point references to the Registry when they are activated and then remove them when they are taken out of service. Consumers can consult the registry in order to determine which end-point reference is needed for the current service task.

Report a bug

14.2. HOW THE REGISTRY WORKS

1. The JBoss Enterprise Service Bus funnels all interaction with the Registry through the registry interface.
2. It then calls a JAXR implementation of this interface.
3. The JAXR API needs to utilize a JAXR implementation. (By default, this is Apache Scout.)
4. Apache Scout, in turn, calls the Registry.

Report a bug

14.3. JUDDI CONSOLE

The jUDDI Console is a web-based graphical interface that you must use in order to configure the jUDDI Registry. It is accessible at http://localhost:8080/uddi-console/.

Report a bug

14.4. GRANT ACCESS TO THE JUDDI CONSOLE

Prerequisites

- A user with the name "root" who has been assigned the security roles of "user" and "admin".

You must log in as a jUDDI Publisher named root to give anyone administration rights. Once a user has these administration rights, they can grant them to other users.

Procedure 14.1. Task

1. Open a web browser session and go to the jUDDI Console at http://localhost:8080/uddi-console/. Log in as root.
2. Click "Publisher".
3. From the Publisher ID list, click on the username.
4. Select the "Is Admin" checkbox.

Result

The user you selected now has administrative rights.

Report a bug

14.5. JUDDI M-BEANS

You can query jUDDI M.-Beans in the JMX console. Doing so allows you to observe Service Registry operations. These are the M.-Beans available:

- org.apache.juddi.api.impl.UDDIServiceCounter
- org.apache.juddi.api.impl.UDDICustodyTransferCounter
- org.apache.juddi.api.impl.UDDIInquiryCounter
- org.apache.juddi.api.impl.UDDIPublicationCounter
- org.apache.juddi.api.impl.UDDISecurityCounter
- org.apache.juddi.api.impl.UDDISubscriptionCounter

Each UDDI operation under the API supplies the following functionality for each method:

- successful queries
- failed queries
- total queries
- processing time
- an aggregate count of total/successful/failed per API

Only one operation is available: resetCounts.

Report a bug

CHAPTER 15. ADMINISTERING YOUR SYSTEM WITH THE JBPM WEB CONSOLE

15.1. JBPM

The JBoss Business Process Manager (jBPM) is a workflow management tool that provides the user with control over business processes and languages. jBPM 3 is used as default.

Report a bug

15.2. JBPM WEB CONSOLE

The jBPM Console is a web-based interface for administering the JBoss Business Process Manager. It is available at http://localhost:8080/jbpm-console/.

Report a bug

CHAPTER 16. ADMINISTERING YOUR SYSTEM WITH THE BPEL WEB CONSOLE

16.1. BPEL WEB CONSOLE

The BPEL Web Console is a tool that allows you to view:

- any process definitions you have deployed to the *BPEL engine*
- the process instances executing in the BPEL engine
- a process' execution history
- the query pertaining to the execution history

Find the console at http://localhost:8080/bpel-console.

IMPORTANT

It is recommended that you only open one BPEL Console window at a time within your browser or you may have trouble logging in or you may see a blank window once you have logged in.

Report a bug

16.2. BUSINESS PROCESS EXECUTION LANGUAGE (BPEL)

Business Process Execution Language (BPEL) is an OASIS-standard language for business rules orchestration. Refer to http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html for more information.

Report a bug

16.3. BUSINESS RULE ORCHESTRATION

Business rule orchestration refers to the act of specifying actions within business processes via web services.

Report a bug

16.4. PROCESS DEFINITION

A BPEL Process Definition is an XML file that acts as a template for a process. when published, the process definition creates a process that is a web service in its own right.

Report a bug

16.5. PROCESS INSTANCE

A process instance is one execution of a process definition.

Report a bug

16.6. VIEW DEPLOYED PROCESSES WITH THE BPEL WEB CONSOLE

Procedure 16.1. Task

1. Launch a web browser and go to http://localhost:8080/bpel-console.
2. Input your user name and password.
3. Click on the **`Manage Instances`** tab to see which BPEL processes are currently deployed. You will also see version information for each of these processes.
4. Select a process definition to open it. In the bottom panel you will see a list of process instances that are active for that particular definition.

NOTE

Only one version of a process can be active at a time. When you open a process definition, the active version is automatically selected.

5. Sometimes you will find that you need to manage a "retired" version (for example, in order to terminate running instances). In these cases, click **`More`** - **`Change Version`** and then select the version you want.

NOTE

If there is no version for a particular process archive, (such as **`Quickstart_bpel_simple_invoke.jar`**), it is treated as version zero. (In this case, **`Quickstart_bpel_simple_invoke-1.jar,`** will be the next version deployed.)

Report a bug

16.7. BUSINESS PROCESS ANALYTICS FORMAT (BPAF)

The Business Process Analytics Format (BPAF) is designed to provide you with information about the efficiency and effectiveness of your organizational processes.

Report a bug

16.8. VIEW BPAF DATA WITH THE BPEL WEB CONSOLE

Procedure 16.2. Task

1. Launch a web browser and go to http://localhost:8080/bpel-console.

2. Input your user name and password.
3. Click on the **`Manage Instances`** tab to see which BPEL processes are currently deployed. You will also see version information for each of these processes.
4. Select a process definition to open it. In the bottom panel you will see a list of process instances that are active for that particular definition.
5. Use the **`Execution History`** to produce a chart. Here you can specify a particular period of time to review and choose whether or not to include failed and terminated instances in the chart.

Report a bug

16.9. LIST OF SHORTCUT KEYS TO USE WHEN NAVIGATING THE EXECUTION HISTORY CHART

Table 16.1. Navigation Keystrokes

Keyboard or Mouse Command	Result
Up Arrow	Zoom In
Down Arrow	Zoom Out
Left Arrow	Half-Page Left
Right Arrow	Half-Page Right
Page-Up	Page Left
Page-Down	Page Right
TAB	Next Focus
Shift-TAB	Previous Focus
HOME	Max Zoom Out
ENTER	Max Zoom In to Focus
Mouse Drag	Scroll Chart
Shift Mouse Drag	Drag Select/Zoom
Mouse Wheel Up/Z	Zoom In
Mouse Wheel Down/X	Zoom Out
Backspace/Back Button	Back

Keyboard or Mouse Command	Result
Right Mouse-Click	Context Menu
Left-Click	Set Focus
Double-Click	Maximise Zoom-In-to-Focus

Report a bug

16.10. ACTIVATE THE BPEL WEB CONSOLE'S LOGGING FUNCTIONALITY

Procedure 16.3. Task

1. Open the **deploy.xml** file in a text editor (for the bpel_helloworld quick start, this would be **vi SOA_ROOT/jboss-as/samples/quickstarts/bpel_hello_world/bpelContent/deploy.xml**
2. Edit the file as follows:

```
<deploy xmlns="http://www.apache.org/ode/schemas/dd/2007/03"
 xmlns:bpl="http://www.jboss.org/bpel/examples"
 xmlns:intf="http://www.jboss.org/bpel/examples/wsdl">

 <process name="bpl:HelloGoodbye">
  <active>true</active>
        <process-events generate="all"/>
  <provide partnerLink="helloGoodbyePartnerLink">
   <service name="intf:HelloGoodbyeService"
port="HelloGoodbyePort"/>
  </provide>
 </process>
</deploy>
```

3. Save the file and exit.
4. Open the **bpel.properties** file in the text editor: **vi SOA_ROOT/jboss-as/server/PROFILE/deploy/riftsaw.sar/bpel.properties**
5. Switch on the process-events option for the particular process you want to log and make sure that org.jboss.soa.bpel.console.bpaf.BPAFLogAdapter is enabled.
6. Save the file and exit.

Report a bug

16.11. VIEW INSTANCE DATA WITH THE BPEL WEB CONSOLE

Procedure 16.4. Task

1. Launch a web browser and go to http://localhost:8080/bpel-console.
2. Input your user name and password.
3. Click on the **Manage Instances** tab to see which BPEL processes are currently deployed. You will also see version information for each of these processes.
4. Select a process definition to open it. In the bottom panel you will see a list of process instances that are active for that particular definition.

NOTE

Only one version of a process can be active at a time. When you open a process definition, the active version is automatically selected.

5. Click the **Instance Data** button.
6. The **View** tab shows the instance execution graph, while the **Source** tab below it shows all of the "activity" events.

Report a bug

16.12. INSTANCE EXECUTION GRAPH

The instance execution graph is a visual representation of a running instance of a process. It tells the user about the process instance's performance over time.

Report a bug

16.13. VIEW THE INSTANCE EXECUTION GRAPH WITH THE BPEL WEB CONSOLE

Procedure 16.5. Task

1. Launch a web browser and go to http://localhost:8080/bpel-console.
2. Input your user name and password.
3. Click on the **Manage Instances** tab to see which BPEL processes are currently deployed. You will also see version information for each of these processes.
4. Select a process definition to open it. In the bottom panel you will see a list of process instances that are active for that particular definition.

NOTE

Only one version of a process can be active at a time. When you open a process definition, the active version is automatically selected.

5. Click on the **`Execution Path`** button to see an instance execution graph for the process.

Report a bug

16.14. VIEW A HISTORY INSTANCE QUERY

Prerequisites

- History logging must be enabled.

Procedure 16.6. Task

1. Log into the BPEL Web Console.
2. Choose a process definition and a process status from the list box.

   You can also optionally choose to input the correlation key, the start time and the end time as search criteria.
3. Go to the History Instances List and double-click on a row. A window will pop up showing you all of the execution events that happened when that process ran.

Report a bug

16.15. ACTIVE PROCESS DEFINITION

When you deploy the first version of a BPEL process definition, it automatically becomes the active definition. If this definition is subsequently changed and redeployed, then that version is "retired", and a new version becomes active automatically.

Report a bug

16.16. RETIRED PROCESS DEFINITION

If the active process definition is changed and redeployed, the old version is "retired". The new version automatically becomes active. The only difference between an active version and a retired one is that a retired one can no longer create new process instances. However, if there are active process instances associated with the retired process version, then these will continue to run.

Report a bug

16.17. MANUALLY RETIRE AN ACTIVE PROCESS DEFINITION

Procedure 16.7. Task

1. Launch a web browser and go to http://localhost:8080/bpel-console.
2. Input your user name and password.

3. Click on the **Runtime** tab.

4. Select the **Deployments** option.

   You will now be able to see the version information and current status (active or retired) of each process definition.

5. Select the particular version of the process definition you want to retire and then press the **Retire** button.

NOTE

If you undeploy a process, its end-points will only deactivate if no previous versions of that process have ever existed.

Report a bug

16.18. END-POINT REFERENCE

An end-point reference (EPR) contains the address information and technical specifications for a service. Indeed, all ESB-aware services are identified using end-point references. It is through these references that services are contacted. The are stored in the registry. Services add their end-point references to the registry when they are launched and should automatically remove them when they terminate. A service may have multiple end-point references. End-point references are also known as binding templates.

End-point references can contain links to the tModels designating the interface specifications for a particular service.

Report a bug

16.19. MANUALLY RE-ACTIVATE A RETIRED PROCESS DEFINITION

Procedure 16.8. Task

1. Launch a web browser and go to http://localhost:8080/bpel-console.

2. Input your user name and password.

3. Click on the **Runtime** tab.

4. Select the **Deployments** option.

   You will now be able to see the version information and current status (active or retired) of each process definition.

5. Select the retired version you want to reactivate and press the **Activate** button (found on the bottom-right of screen.)

Report a bug

16.20. ENABLE UTF-8 SUPPORT FOR PROCESSES OR EXTERNAL WEB SERVICES

Procedure 16.9. Task

1. Check your database to make sure UTF-8 encoding is being used by default.
2. Launch a text editor and open the database's configuration file.
3. Add these settings to the file:

```
hibernate.connection.useUnicode=true
hibernate.connection.characterEncoding=UTF-8
```

4. Save the file and exit.

Report a bug

PART IV. MANAGING MULTIPLE SERVER CONFIGURATIONS

CHAPTER 17. RUNNING MULTIPLE JBOSS ENTERPRISE SOA PLATFORM INSTANCES SIDE-BY-SIDE

17.1. RUNNING APPLICATION SERVERS SIDE-BY-SIDE

Introduction

The JBoss Enterprise SOA Platform can be made to run alongside another JBoss product such as the JBoss Enterprise Application Platform. There are two ways of achieving this:

- by using multi-homing
- by using the Service Bindings Manager

WARNING

If you simultaneously launch multiple servers that each have the same database configuration, you may encounter errors. The server initialises a new database when it is launched with it for the first time. The problems can occur if more than one server attempts to do this simultaneously. To avoid this problem, launch one instance and wait until it has finished initializing the databases before starting the other instances. This only has to be done the first time that the servers are started.

Report a bug

17.2. RUN APPLICATION SERVERS SIDE-BY-SIDE USING MULTI-HOMING

Procedure 17.1. Task

1. Configure your operating system's network interface so it is assigned multiple IP addresses. (Refer to your operating system's documentation for instructions on doing this).
2. Launch each server instance using the **-b** switch to bind all of them to a single IP address. Here is an example: **SOA_ROOT/jboss-as/bin/./run.sh -b 10.34.5.2**

Report a bug

CHAPTER 18. MANAGING YOUR CLUSTER

18.1. CLUSTER

A cluster is a group of loosely-connected computers that work together in such a way that they can be viewed as a single system. The components (or nodes) of a cluster are usually connected to each other through fast local area networks, each node running its own operating system. Clustering middleware orchestrates collaboration between the nodes, allowing users and applications to treat the cluster as a single processor. Clusters are effective for scalable enterprise applications, since performance is improved by adding more nodes as required. Furthermore, if at anytime one node fails, others can take on its load.

Report a bug

18.2. STATELESS SERVICE FAILOVER

The JBoss Enterprise SOA Platform provides "failover" capabilities for stateless services. If one node fails, the service will be resumed on another. The ServiceInvoker hides much of the fail-over complexity from users but it only works with native Enterprise Service Bus messages and, furthermore, not every gateway has been programmed to take advantage of it. Hence, non-ESB Aware messages sent to these gateway implementations may not be able to use service fail-over.

Report a bug

18.3. SERVICEINVOKER

The ServiceInvoker (**`org.jboss.soa.esb.client.ServiceInvoker`**) manages the delivery of messages to the specified Services. It also manages the loading of end-point references and the selection of couriers, thereby providing a unified interface for message delivery.

The ServiceInvoker was introduced to help simplify the development effort as it hides much in the way of the lower-level details and works opaquely with the stateless service fail- over mechanisms. As such, ServiceInvoker is the recommended client-side interface for using services within the JBoss Enterprise SOA Platform.

You can create an instance of the ServiceInvoker for each service with which the client interacts. Once created, an instance examines the registry to determine the primary end-point reference and, in the case of fail-overs, any alternative end-point references.

Report a bug

18.4. LOAD BALANCING

Load balancing is a computer networking methodology to distribute workload across multiple computers or a computer cluster, network links, central processing units, disk drives, or other resources, to achieve optimal resource utilization, maximize throughput, minimize response time, and avoid overload. Using multiple components with load balancing, instead of a single component, also increases reliability through redundancy.

Report a bug

18.5. CONFIGURE A LOAD-BALANCING POLICY

Procedure 18.1. Task

1. Open the global configuration file in a text editor: **`vi SOA_ROOT/jboss-as/server/PROFILE/deployers/esb.deployers/jbossesb-properties.xml`**.
2. Scroll down to the org.jboss.soa.esb.loadbalancer.policy property. Set it with the policy you wish to use.
3. Save the file and exit.

Report a bug

18.6. LOAD BALANCING POLICIES

Table 18.1. Load balancing Policies Available

Policy Name	Description
first available	If a healthy service binding is found it will be used until it dies. The next end-point reference in the list will then be used. There is no load balancing between the two service instances with this policy.
round robin	A standard load-balancing policy whereby each end-point reference is utilised in list order.
random robin	This is like the round robin, but the selection is randomized.

NOTE

The end-point reference list used by the policy may become smaller over time as "dead" EPRs are removed. When the list is exhausted or the time-to-live of the list cache is exceeded, the ServiceInvoker will obtain a fresh list of EPRs from the Registry.

Report a bug

18.7. CHANGE THE REGISTRY'S CACHE'S LIFESPAN

Procedure 18.2. Task

1. Open the global configuration file in a text editor: **`vi SOA_ROOT/jboss-as/server/PROFILE/deployers/esb.deployers/jbossesb-properties.xml`**.

2. Scroll down to the section that contains property name="org.jboss.soa.esb.registry.cache.validityPeriod. Set this property (which is the time-out value) to what you require (the default is sixty seconds):

```
<properties name="core">
<property name="org.jboss.soa.esb.registry.cache.life"
value="60000"/>
<!-- 60 seconds is the default -->
</properties>
```

3. Save the file and exit.

Result

The ServiceInvoker will obtain a fresh list of end-point references from the registry when this time value is exceeded.

Report a bug

18.8. RUN THE SAME SERVICE ON MORE THAN ONE NODE IN A CLUSTER

Procedure 18.3. Task

- To run the same service on more than one node in a cluster, wait until the Registry's cache revalidates.

Report a bug

18.9. REMOVE FAILED END-POINT REFERENCES FROM THE REGISTRY

Procedure 18.4. Task

1. Open the **`jbossesb-properties.xml`** in a text editor: **`vi SOA_ROOT/jboss-as/server/PROFILE/deployers/esb.deployers/jbossesb-properties.xml`**.

2. Scroll down to the section that contains org.jboss.soa.esb.failure.detect.removeDeadEPR. Set this property to true.

3. Save the file and exit.

WARNING

Note that the default setting is false because this feature should be used with extreme care. If it is employed, the end-point reference for a service that is simply overloaded and, therefore, slow to respond may, inadvertently, be removed by mistake. There will be no further interactions with these "orphaned" services you may have to restart them.

Report a bug

18.10. SUPPORT FOR CLUSTERING IN THE BPEL ENGINE

In order to make the BPEL Engine work correctly in a clustered environment, you must configure it to utilise a database shared between all of the environment's nodes. (This is because BPEL persists every process states in a database.)

In order for it to dispatch the SOAP requests to all of the nodes correctly, a *load balancer* is also required.

Report a bug

18.11. CONFIGURE BPEL CLUSTERING

Procedure 18.5. Task

1. Open the **`jboss-beans.xml`** example file in your text editor.
2. Set the @database@ property to one of the following:
   - mysql
   - postgre
   - db2
   - sqlserver
   - oracle
   - sybase
3. Save the file and exit.
4. Copy the **`jboss-beans.xml`** file to **`SOA_ROOT/jboss-as/server/PROFILE/deploy/riftsaw.sar/META-INF/`**.

5. Replace **`SOA_ROOT/jboss-as/server/PROFILE/deploy/cluster/jboss-cache-manager.sar/META-INF/jboss-cache-manager-jboss-beans.xml`** with **`riftsaw-cache-manager-jboss-beans.xml`**.

WARNING

Attempting to install another BPEL Engine deployment can break the integration.

NOTE

If you want to use the service that you have deployed onto the cluster, specify the load balancer's URL instead of the SOAP address in the WSDL file.

Report a bug

18.12. DEPLOY A BPEL PROCESS ON A CLUSTER

Procedure 18.6. Task

- Copy your BPEL artifact into the farm directory: **`cp FILENAME.jar SOA_ROOT/jboss-as/server/PROFILE/farm`**.

NOTE

Remember that clustering is only available for the "production" and "all" profiles.

NOTE

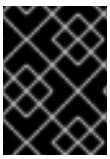
When you invoke your BPEL service, specify the load balancer's URL (instead of the SOAP address specified in the WSDL). The load balancer will then decide which of the cluster's servers to use.

Report a bug

PART V. MANAGING SERVICES

CHAPTER 19. PUBLISHING CONTRACTS

19.1. SERVICE LIST APPLICATION



IMPORTANT

Red Hat is offering the service list functionality as a *Technical Preview* only at this point in time. It will be replaced by different technology in a later release.

The Service List Application is a tool that allows the user to see information about end-points. (You will often need this information if you are utilizing web service end-points exposed by the SOAPProcessor action). The tool is at http://localhost:8080/contract/. The tool groups end-points under the services with which they are associated.

Report a bug

19.2. END-POINT CONTRACT

An end-point contract specifies what an end-point will communicate to other services.

Report a bug

19.3. HOW THE JBOSS ENTERPRISE SOA PLATFORM DISCOVERS END-POINT CONTRACTS

The JBoss Enterprise SOA Platform discovers end-point contracts via looking in the action pipeline for the first action that can publish contract information.

If none of the actions can do so, then the Service List Application displays this message:

```
Unavailable on Contract
```

Report a bug

19.4. PUBLISH A CONTRACT

Procedure 19.1. Task

1. In order to publish contract information, you must give an action the following **`org.jboss.internal.soa.esb.publish.Publish`** annotation. (This example uses the SOAPProcessor for demonstrative purposes):

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

2. Implement the **`org.jboss.soa.esb.actions.soap.ContractPublisher`** interface (You only need to implement one method):

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

Report a bug

CHAPTER 20. DEPLOY ARCHIVE FILES

20.1. HOT DEPLOYMENT

"Hot deployment" refers to the SOA Platform server's ability to detect a deployed ESB archive as soon as it lands in the server profile's deploy directory. The SOA Platform server software constantly monitors that directory and automatically detects any changes to it. It can also detect changes to the state of existing archives and automatically re-deploy them.

These actions have *life-cycle support*. This means that, upon hot re-deployment, they terminate "gracefully" by finishing active requests. They will not accept any more incoming messages until they are re-started. (All of this occurs automatically.)

NOTE

You cannot deploy archives if the JBoss Enterprise SOA Platform is running in standalone mode.

The node monitors the time-stamp on this file and re-reads its contents if a change occurs.

Report a bug

20.2. HOT DEPLOYMENT AND JBOSSESB.SAR

The jbossesb.sar archive can be deployed live. It will deploy when:

- its time-stamp changes, (if the archive is compressed.)
- the timestamp of the **`META-INF/jboss-service.xml`** file changes, (if the archive is in exploded form.)

Report a bug

20.3. HOT DEPLOYMENT AND ESB ARCHIVES

An **`*.esb`** archive will automatically redeploy when:

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

Report a bug

20.4. REDEPLOY A RULES FILE

Procedure 20.1. Task

1. To redeploy a .DRL or .DSL file, redeploy the jbrules.esb directory by copying it back into **`SOA_ROOT/jboss-as/server/PROFILE/deploy`**.

2. Alternatively, you can activate the Action Configuration's ruleReload feature. After activating this functionality, if a rule file changes, it is re-loaded automatically.

Report a bug

20.5. REDEPLOY A TRANSFORMATION FILE

Procedure 20.2. Task

1. Redeploy the .ESB archive in which it resides by copying it back into the **`SOA_ROOT/jboss-as/server/PROFILE/deploy`** directory.
2. Alternatively, launch a web browser and log into the Monitoring and Management Console at http://localhost:8080/admin-console.
3. Send out a notification message over the Java Message Service. Smooks will process this event causing it to reload automatically.

Report a bug

20.6. REDEPLOY A BUSINESS PROCESS DEFINITION

Prerequisites

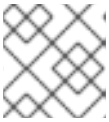
- JBoss Developer Studio

Procedure 20.3. Task

- Use the jBPM Eclipse plug-in to deploy a new version of a business process definition to the jBPM database.

NOTE

Please be aware that only a fresh process instance will use this new version. Existing process life-cycles will still use the previous definition.



NOTE

Note that this procedure works in standalone mode, too.

Report a bug

20.7. RELOAD RULES WHILST RUNNING IN STANDALONE MODE

Procedure 20.4. Task

- Run **`ruleReload`**.

Report a bug

CHAPTER 21. INTEGRATING EXTERNAL WEB SERVICES WITH THE JBOSS ENTERPRISE SOA PLATFORM

21.1. WEB SERVICE

A web service is a way of making two applications communicate over the web. A web service consists of a set of tools to achieve this aim. There are two types of web service: REST-compliant ones, (the purpose of which is to manipulate XML representations of web resources) and arbitrary Web services (through which the service can expose any operation).

Report a bug

21.2. WEB SERVICE END-POINT

A web service end-point is software that implements a web service. They are used to implement message-based communication between web services in a service-oriented architectural environment.

The external applications to which these registry entries point can include .NET programs, other external Java-based application servers and LAMP software bundles.

Report a bug

21.3. WEB SERVICES DESCRIPTION LANGUAGE (WSDL)

The Web Services Description Language (WSDL) is an XML-based language that is used to define Web service interfaces. An application that consumes a Web service parses the service's WSDL document to discover the:

- location of the service
- the operations that the service supports
- the protocol bindings the service supports (SOAP, HTTP, etc)
- access procedure

For each operation, the WSDL describes the interface format to which the client must adhere.

Report a bug

21.4. REST

REST (Representational State Transfer) is a type of software architecture, consisting of clients and servers. It was designed specifically for distributed networks, including the internet. Central to this is the idea of multiple resources (data sources), each of which is identified by a unique, global address. In order to manipulate these resources,the clients and servers communicate via a standardized interface and exchange data.

Report a bug

21.5. SOAPPROCESSOR

The SOAPProcessor is an action that allows you to invoke a JBossWS-hosted web service end-point through any JBossESB-hosted listener. Via this action, the enterprise service bus can expose web service end-points for services that do not already expose them. You can then invoke services over any transport channel supported by the enterprise service bus, such as HTTP, FTP or JMS.

The SOAPProcessor supports both the JBossWS-Native and JBossWS-CXF stacks.

Report a bug

21.6. SOAPPROXY

The SOAPProxy is an action that "consumes" external web service end-points. It also allows you to re-publish a web service end-point via the Enterprise Service Bus. Sitting between the external services and the ESB, the purpose of this intermediary is to provide an abstraction layer it provides the following functionality:

- it facilitates loose coupling between the client and service (since they are both completely unaware of each other.)
- it means the client no longer has a direct connection to the remote service's hostname/IP address.
- the client will see modified WSDL that changes the inbound/outbound parameters. At a minimum, the WSDL must be tweaked so that the client is pointed to the ESB's exposed end-point instead of the original, now proxied endpoint.
- it allows you to introduce a transformation of the SOAP envelope/body via the action pipeline both for the inbound request and outbound response.
- it makes service versioning possible since clients can connect to two or more proxy end-points on the enterprise service bus, each with its own WSDL and/or transformations and routing requirements, and the ESB will send the appropriate message to the appropriate endpoint and provide an ultimate response.
- it allows for complex context-based routing via ContentBasedRouter.

Report a bug

21.7. ADVANTAGES OF INTEGRATING WEB SERVICES WITH THE ENTERPRISE SERVICE BUS

The JBoss Enterprise SOA Platform's enterprise service bus sits between the producer of the resource and the client application that ultimately consumes it. The ESB's purpose is to provide an abstraction layer, thereby giving these advantages:

- the client and the service can be coupled loosely since they will be completely unaware of each other's existence.

- the client is no longer connected directly to the remote service's hostname or IP address.
- the client can see a modified WSDL, changing the inbound/outbound parameters. (Note that, at a minimum, one must modify the WSDL so that the client is pointing to the end-point exposed by the Enterprise Service Bus, rather than the original end-point.)
- you can introduce a SOAP envelope/body transformation via the action pipeline that will apply to both the inbound request and the outbound response.
- you can implement service versioning since clients can connect to two or more proxy end-points, each with its own WSDL and/or transformations and routing requirements. The Enterprise Service Bus will send the appropriate message to the correct end-point and then return a response.
- the **`ContentBasedRouter`** class can be used to introduce advanced routing functionality.

Report a bug

21.8. CONFIGURE WEB SERVICE INTEGRATION

Procedure 21.1. Task

- QE/SME to provide information.

Report a bug

21.9. REPUBLISH A WEB SERVICE USING THE SOAPPROXY ACTION

Procedure 21.2. Task

- QE/SME to provide information.

Report a bug

21.10. CONTENT-BASED ROUTER

Content-based routers send messages that do not have destination addresses to their correct end-points. Content-based routing works by applying a set of rules (which can be defined within XPath or Drools notation) to the body of the message. These rules ascertain which parties are interested in the message. This means the sending application does not have to supply a destination address.

A typical use case is to serve priority messages in a high priority queue. The advantage here is that the routing rules can be changed on-the-fly while the service runs if it is configured in that way. (However, this has significant performance drawbacks.)

Other situations in which a content-based router might be useful include when the original destination no longer exists, the service has moved or the application simply wants to have more control over where messages go based on its content of factors such as the time of day.

Report a bug

21.11. STATIC-BASED ROUTER

A static-based router helps to coordinate information across your network. It transfers information between servers and tells them where to send their messages. You can program it to take certain routes if you feel the default is inefficient. You can only use it to route to other services.

Report a bug

21.12. ROUTING KEY

A routing key is used to match messages and the items to which they are bound. In the context of the JBoss Enterprise SOA Platform, the routing key can be applied to a message which will then be sent to a queue which has been given the same routing key.

Report a bug

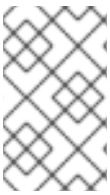
PART VI. AUDITING AND TROUBLESHOOTING YOUR SYSTEM

CHAPTER 22. SYSTEM AUDITING

22.1. MESSAGE STORE

The message store is a database persistence mechanism that has been designed to allow you to do audit-tracking. The message store reads and writes messages upon request. Each message must have a unique identification number. As with other ESB services, the message store is "pluggable", which means that you can "plug in" your own persistence mechanism should you desire to do so, though a default database persistence mechanism is supplied.

In the event of a system failure, the message store is also used as a holding place for messages that need to be re-delivered.



NOTE

If something other than a database is required, such as a file persistence mechanism, you can write your own service and then override the default behaviour with a configuration change.

Report a bug

22.2. SERVICE ROUTE FILTER

The ServiceRouteFilter (**`org.jboss.internal.soa.esb.message.filter.ServiceRouteFilter`**) is an auditing mechanism that allows you to track a message's path through different services. Like any other filter, you need to enable it from within the **`jbossesb-properties.xml`** file.

Report a bug

22.3. AUDIT THE DATA IN THE MESSAGE STORE

Procedure 22.1. Task

1. Open **`jbossesb-properties.xml`** in a text editor: **`vi jbossesb-properties.xml`**
2. Go to the section called filters and edit it as per the following code sample:

```
        @lt;properties name="filters"@gt;
        @lt;property name="org.jboss.soa.esb.filter.1"
value="org.jboss.internal.soa.esb.message.filter.MetaDataFilter"/@gt
;
        @lt;property name="org.jboss.soa.esb.filter.2"
value="org.jboss.internal.soa.esb.message.filter.GatewayFilter"/@gt;
        @lt;property name="org.jboss.soa.esb.filter.3"
value="org.jboss.internal.soa.esb.message.filter.ServiceRouteFilter"
/@gt;
    @lt;/properties@gt;
```

3. Save the file and exit.

it will now check messages and services for whether or not service route information should be recorded. On either a per-service or a per-message level, you can tell the filter to add the route information into the context.

4. To configure it on a service level, add recordRoute="true" in your service definition.

```
<service
        category="FirstServiceESB"
        name="SimpleListener"
        description="Hello World"
        recordRoute="true">
```

5. To configure it on a message level, add a service-record-route property to the message properties and set it to **`true`**.

Report a bug

22.4. TRACEFILTER

The TraceFilter (**`org.jboss.internal.soa.esb.message.filter.TraceFilter`**) is the JBoss Enterprise SOA Platform's meta-data filter. Its role is to record entries in the log whenever a message interacts with a component. It enables you to trace an event and have information on it returned to you. For example, you can set it to trace certain kinds of messages and display their movements to make it easier to monitor them.

Report a bug

22.5. LOG MESSAGE

By default, every interaction between components and messages is logged. Log messages contain header information linking them to the main messages. Thus, you can correlate messages sent across multiple SOA instances.

Report a bug

22.6. IDENTIFY A LOG MESSAGE

Procedure 22.2. Task

- **Determining if a Message is a Log Messages**
  To identify a log message, open it up and see if it adheres to the following format:

```
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 ]
```

Report a bug

22.7. FILTER FOR LOG MESSAGES

Procedure 22.3. Task

1. **Open the jbossesb-properties.xml File**
   Open the **`jbossesb-properties.xml`** in a text editor: **`vi SOA_ROOT/jboss-as/server/PROFILE/deployers/esb.deployers/jbossesb-properties.xml`**.

2. Scroll down to the "Filter" section of the file.

3. Set the org.jboss.soa.esb.messagetrace property to on. Now that it is enabled, every message that passes through it is logged.

4. To gain more precise control over which messages are logged and which are ignored, set the org.jboss.soa.esb.permessagetrace property to on as well. This causes the filter to ignore those messages for which the org.jboss.soa.esb.message.unloggable property is set to yes.

5. **Save**
   Save the file and exit.

Result

The TraceFilter is switched on. Whenever a message passes through this filter, you will see the following at the information level:

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

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

NOTE

This filter does not affect the input or output message.

Report a bug

CHAPTER 23. TROUBLESHOOTING

23.1. TROUBLESHOOTING YOUR JBOSS ENTERPRISE SOA PLATFORM INSTALLATION

Here are the solutions to some of the problems users most commonly encounter.

JBOSS_HOME set incorrectly

If the optional environmental variable, JBOSS_HOME, is set then it must point to the correct directory. If you have multiple installations, check that it is pointing to the one that you are trying to run.

WARNING

Do not set this variable unless you have a specific need to do so.

Java installed incorrectly

If the Java environment has been installed or configured incorrectly, then the JBoss Enterprise SOA Platform will not function.

VM Cannot Allocate Sufficient Memory

This error occurs when there is not enough free memory available to the system to satisfy the JBoss Enterprise SOA Platform';s requirements. You can increase the amount available in one of three ways: by exiting applications, allocating more virtual memory, or physically increasing the amount of RAM installed on the system.

Report a bug

23.2. TROUBLESHOOTING THE BOOT PROCESS

Errors in the server.log are indicated by the keyword "ERROR". If you see an error in the log, look through this list to find the cause:

1. "Address already in use" - There is already a server running on port 8080.
2. "Java not found" - The Java JRE may not be installed, or if it is, your PATH environment variable is not set to locate the java runtime.
3. "Class not found" - The CLASSPATH environment variable is not set properly. You really don't need to set this variable as the server startup script sets it for you.
4. If you see any of these errors, examine the server.log messages that come before and after the error message for additional information regarding the root cause of the error.

Report a bug

23.3. END-POINT REFERENCE

An end-point reference (EPR) contains the address information and technical specifications for a service. Indeed, all ESB-aware services are identified using end-point references. It is through these references that services are contacted. The are stored in the registry. Services add their end-point references to the registry when they are launched and should automatically remove them when they terminate. A service may have multiple end-point references. End-point references are also known as binding templates.

End-point references can contain links to the tModels designating the interface specifications for a particular service.

Report a bug

23.4. TROUBLESHOOTING REGISTRY SERVICES

Although the Registry lists services, it cannot determine their statuses. In other words, there can be no guarantee that an end-point reference listed in the Registry is valid (it may, for instance, be malformed or it may be representing a service that is no longer active.) This is because the JBoss Enterprise SOA Platform does not currently possess life-cycle monitoring functionality. Hence, the administrator must manually update or remove invalid end-point references.

IMPORTANT

ESB services create their own end-point references automatically. These end-points are internal implementations, so you should never modify them. If you do, Red Hat will not support you.

Report a bug

23.5. REMOVE AN END-POINT REFERENCE FROM THE REGISTRY

Prerequisites

- Ensure the system is in an inactive state

Procedure 23.1. Task

1. Open the end-point reference file in your text editor.
2. Set the end-point reference's remove-old-service tag value to true:

```
<jms-listener name="JMS-ESBListener"
busidref="quickstartEsbChannel">
        <property name="remove-old-service" value="true"/>
</jms-listener>
```

WARNING

Always use this option with caution, because the entire service, including every one of its end-point references, will be removed.

3. Save the file and exit.

Report a bug

23.6. APACHE SCOUT

Apache Scout is an open source implementation of JAXR, created by the Apache Project.

There are currently four implementations of the **`org.jboss.soa.esb.scout.proxy.transportClass`** class, one each for SOAP, SAAJ, RMI and Embedded Java (Local).

Report a bug

23.7. SERVICE REGISTRY AND APACHE SCOUT TROUBLESHOOTING CHECKLIST

- If you decide to use remote method invocation, be sure to obtain the **`juddi-client.jar`** file, (**`SOA_ROOT/jboss-as./server/PROFILE/deployers/esb.deployer/lib/juddi-client-VERSION.jar`**)
- Ensure that the **`jbossesb-properties.xml`** file is on the class-path and that it is being read correctly. If not, the Registry try to use "null" as the name with which to instantiate classes.
- Make sure that **`META-INF/esb.juddi.client.xml`** file specifies a valid transport.
- Make sure that the **`persistence.xml`** file's settings are valid and that the **Hibernate** dialect you have chosen matches that used by the database.
- Ensure that the **`esb.juddi.xml`** file is on the class-path. This contains some of the Registry's configuration settings.
- Sometimes, if a service fails or does not shut down cleanly, old entries may linger on in the Registry. Remove these manually.

Report a bug

23.8. FURTHER SERVICE REGISTRY TROUBLESHOOTING RESOURCES

To learn more about troubleshooting the Registry, visit these locations:

- The JBoss jUDDI Wiki: http://www.jboss.org/community/docs/DOC-11217
- The JBoss ESB User Forum: http://community.jboss.org/en/jbossesb?view=discussions.

Report a bug

23.9. JAVA MESSAGE SERVICE

A Java Message Service (JMS) is a Java API for sending messages between two clients. It allows the different components of a distributed application to communicate with each other and thereby allows them to be loosely coupled and asynchronous. There are many different Java Message Service providers available. Red Hat recommends using HornetQ.

Report a bug

23.10. IBM WEBSPHERE MQ JAVA MESSAGE SERVICE PROVIDER DIAGNOSTIC TRACING FUNCTIONALITY

The IBM WebSphere MQ Java Message Service component has a message tracing capability. This can be useful when you are trying to diagnose problems.

Report a bug

23.11. DIAGNOSTIC TRACE

A diagnostic trace is a method of troubleshooting where you can trace files and settings to see if there has been an error in the production process.

Report a bug

23.12. ENABLE DIAGNOSTIC TRACING FOR THE IBM WEBSPHERE MQ JCA ADAPTER

You can set the diagnostic trace as a property on the resource adapter, or in the Java Virtual Machine's system properties. With the JBoss ESB/Application, Red Hat recommends you use the JVM system properties approach.

However, on systems that are started via use of the **`./run.sh`** shell script, you should use the following approach:

Procedure 23.2. Task

1. **Open the run.conf File**
   Open the file in a text editor: **`vi SOA_ROOT/jboss-as/bin/run.conf`**.

2. **Edit the run.conf File**

Appending the following lines 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"
```

3. **Enable Client Logging**
   Still in the text editor, set the 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"
```

4. **Save**
   Save the file and exit.

Report a bug

23.13. ENABLE DIAGNOSTIC TRACING FOR THE IBM WEBSPHERE MQ JAVA CLIENT

Procedure 23.3. Task

- **Call the enableTrace Static Method**
  Call the **`com.ibm.mq.MQEnvironment`**'s enableTrace static method.

Report a bug

PART VII. PERFORMANCE TUNING

CHAPTER 24. PERFORMANCE TUNING

24.1. PERFORMANCE TUNING

Performance tuning is the process of fine-tuning and improving your system's performance. This may be through configuring the settings to process more data or enacting load-bearing capabilities for faster performance.

Report a bug

24.2. TUNE THE JBOSS ENTERPRISE SOA PLATFORM FOR HIGH PERFORMANCE

Procedure 24.1. Task

- **Learn How to Tune the Product**
  To learn about performance tuning, go to this website:
  http://community.jboss.org/wiki/JBossESBPerformanceTuning.

Report a bug

24.3. REGISTRY PERFORMANCE

In order to improve performance and reliability of your registry, Red Hat recommends that you replicate or federate it. In this way, you will prevent it being a single point of failure.

Report a bug

24.4. JMS MESSAGE PRIORITIZATION

JMS message prioritization is a feature that allows you to dictate which messages have priority when being processed by the enterprise service bus. Configure the priority on the gateway. It will then be passed onto the ESB services which are subsequently invoked. You should set a priority for any JMS transports which are used to communicate with ESB services.

The message flow priority is passed through ESB invocations into the invoked services as part of the message context. However it is not present within the message while the service is being executed. Rather, the priority will be held within a thread local context so that it can be propagated through any subsequent invocations of the ServiceInvoker.

Report a bug

24.5. CONFIGURE THE PRIORITY OF JMS MESSAGES

Procedure 24.2. Task

1. Open the gateway's configuration file in a text editor.

2. Add the following code to either the listener, bus or provider area of the file:

```
<property name="messageFlowPriority" value="X"/>
```

The value of x can be a number from **0** to **9** inclusive, where **0** is the lowest priority and **9** is the highest.

3. Save the file and exit.

Report a bug

24.6. GATEWAYS ON WHICH PRIORITIZATION CAN BE SET

You can set JMS message prioritization on the following gateways:

- Scheduled (including file and so forth)
- Groovy
- JMS (If this transport is invoked through the JMS Courier, then the priority will also be used to configure the MessageProducer.)
- SQL
- JCA inflow
- Camel
- Hibernate
- Http
- JBoss Remoting
- UDP
- EBWS (For EBWS the property must be specified within the configuration file's "service" element.)

Report a bug

24.7. DYNAMIC CONFIGURATION OF THE MESSAGEAWARELISTENER THREAD POOL

The number of message threads in the MessageAwareListener thread pool changes dynamically. This means that the minimum and maximum number of threads alter as needed. This functionality is exposed through the service M-Bean.

Report a bug

APPENDIX A. SOME USEFUL DEFINITIONS

A.1. SERVICE

A service is a list of action classes that process an ESB Message in a sequential manner. Each service element consists of one or more listeners and one or more actions. These are set within the **`jboss-esb.xml`** configuration file.

Report a bug

A.2. BOOT-STRAPPER MODE

Putting your software into boot-strapper mode tells it what to load and when to do so.

Report a bug

A.3. MESSAGE RE-DELIVERY SERVICE

The Message Re-delivery Service attempts to redeliver messages when none of the end-point references work.

Report a bug

A.4. ACTION PIPELINE

The action pipeline consists of a list of action classes through which messages are processed. Use it to specify which actions are to be undertaken when processing the message. Actions can transform messages and apply business logic to them. Each action passes the message on to the next one in the pipeline or, at the conclusion of the process, directs it to the end-point listener specified in the ReplyTo address.

The action pipeline works in two stages: normal processing followed by outcome processing. In the first stage, the pipeline calls the process method(s) on each action (by default it is called "process") in sequence until the end of the pipeline has been reached or an error occurs. At this point the pipeline reverses (the second stage) and calls the outcome method on each preceding action (by default it is processException or processSuccess). It starts with the current action (the final one on success or the one which raised the exception) and travels backwards until it has reached the start of the pipeline.

Report a bug

A.5. RUN.SH

`run.sh` is the shell script the user runs to launch the JBoss Enterprise SOA Platform. The Microsoft Windows equivalent is **`run.bat`**. The script contains the commands needed to start the server with the profile and port binding which the user has specified in the shell. The script is found in the **`SOA_ROOT/jboss-as/bin`** directory.

Report a bug

A.6. CLASS-PATH

A classpath is a setting that tells the Java Virtual Machine where, on the filesystem, to find user-created classes and packages.

Report a bug

A.7. BUSINESS PROCESS DEFINITION

A business process definition determines the common elements of any runtime instances being used in a process. It is reusable.

Report a bug

A.8. SERVER PROFILES

A server profile is a set of pre-determined settings for running the JBoss Enterprise SOA Platform in different ways. The following profiles come with the product: all, default, minimal, production, standard and web. They are found in the **`SOA_ROOT/jboss-as/server/`** directory. The user specifies which profile to run when launching the software by using the **`-c`** switch. If none is specified, the "Default" profile is used.

Report a bug

A.9. DATASOURCE NAME

A datasource name (DSN) is the title given to a particular piece of data. For example, a DSN could refer to the name of a database.

Report a bug

A.10. DECISION TABLE

A decision table contains a list of actions. These are undertaken by the system when required.

Report a bug

A.11. STATELESS SERVICE

A stateless service is a self-contained service that independently performs tasks instead of having to receive instructions from the user. Additionally, it does not need to use up vast amounts of data to identify objects.

Report a bug

A.12. SERVICE BINDING

A service binding allows you to transport data between clients and services by linking them.

Report a bug

A.13. ENTERPRISE JAVA BEAN

An Enterprise Java Bean is a Java component architecture designed for enterprise applications. It can be used to create these applications and then deploy them to a server.

Report a bug

A.14. LOOSE COUPLING

Loose coupling is when two components are linked together to perform certain tasks, but remained unlinked the rest of the time.

Report a bug

A.15. PERSISTENCE MECHANISM

A persistence mechanism is a fail-over property. It makes an object persistent, meaning it can automatically start up again after a shutdown and resume the task it was previously performing.

Report a bug

A.16. RESOURCE ADAPTER

A resource adapter allows you to modify an application so that other components can be "plugged" into it. These components are then able to communicate with the rest of the system using the adapter.

Report a bug

A.17. SHELL SCRIPT

A shell script is a text file containing a series commands for UNIX-based operating systems like Red Hat Enterprise Linux. They call on the shell (terminal) when they run. The Microsoft Windows equivalent is a batch file.

Report a bug

A.18. WEB CONTAINER

A web container works with Java servlets. It is responsible for managing their performance as well as making sure they are sending and receiving the correct information. The JBoss Enterprise Application Platform is a type of web container.

Report a bug

A.19. INITIAL CONTEXT FACTORY

An initial context factory is where initial context objects are created. These objects are used to create and view naming and directory properties.

Report a bug

A.20. USERNAMETOKEN

The UsernameToken is used to "propagate" a username (and optionally a password) throughout a system to avoid having to login multiple times in a single session.

Report a bug

A.21. SCHEMA VALIDATION

This is the process of validating or "checking" code to make sure it works. You can make sure there are no errors in your XML code by running it through a schema validation.

Report a bug

A.22. BYTE ARRAY

As the name suggests, this is an array of bytes that make up objects like memory. You can create an array of bytes for use with message sending and processing packets.

Report a bug

A.23. EXTENDED TRANSACTIONAL CLIENT

An extended transactional client allows you to send and receive messages from local queue managers. It also allows you to view and update external queue managers.

Report a bug

A.24. CONNECTION POOLING

Connection pooling is a back-end way of connecting a server to multiple clients. Once a connection pool is created, an application server can draw on it to perform stored actions (for example, requesting a

database to do a certain task). It simplifies common tasks as the actions are ready to go in the connection pool for when the user decides to deploy them.

Report a bug

A.25. POOLED DATABASE MANAGER

As the name implies, this manager works with pooled databases and allows for them to be accessed, managed and configured efficiently.

Report a bug

A.26. CIPHER TRANSFORMATION

Use this transformation to decrypt information.

Report a bug

A.27. CONCURRENCY CONTROL

This method of control allows multiple operations to run concurrently while making sure all their processes are running correctly and efficiently.

Report a bug

A.28. UNIFORM RESOURCE IDENTIFIER

A uniform resource identifier (URI) uses a sequence of alphanumeric characters to identify a resource in the system. A web URL is one type of URI.

Report a bug

A.29. PROVIDER ADAPTER

A provider adapter allows applications to receive information from remote providers.

Report a bug

A.30. IMPLEMENTATION CLASS

An implementation class defines how an object which belongs to certain classes is implemented.

Report a bug

A.31. INTERCEPTOR CLASS

An interceptor class is applied to an object to make it perform additional actions that have been defined in the class.

Report a bug

A.32. TRANSACTED FLAG

You can set your session to have a transacted flag with the value of true or false. It can be set to true on specific endpoints to make them transactional. This means that all the actions of the endpoint can be grouped into one singular action instead of lots of smaller ones.

Report a bug

A.33. JAVA CONNECTOR ARCHITECTURE (JCA) TRANSPORT

The Java Connector Architecture (JCA) Transport is a Java-based piece of architecture that works as a service integrator. It is a connector that links application servers and enterprise information systems.

Report a bug

A.34. JCA BRIDGE

The JCA bridge is a dispatcher which can open and close connections. It identifies connections set by the user and can detect connectors and gateways.

Report a bug

A.35. JCA ADAPTER

The JCA adapter acts as a "go between" that links application servers and enterprise information systems.

Report a bug

A.36. END-POINT CLASS

An end-point class lets you identify resources and services on your network by providing their network address.

Report a bug

A.37. SERVICE PROVIDER

A service provider gives access to services, creates descriptions of them and publishes them to the service broker.

Report a bug

A.38. SERVICE BROKER

A service broker hosts the registry of service descriptions. It is responsible for linking a service requester to a service provider.

Report a bug

A.39. SERVICE REQUESTER

A service requester is responsible for discovering a service. It does so by searching through the service descriptions given to it by the service broker. A requester is also responsible for binding together services obtained from the service provider.

Report a bug

A.40. MESSAGING QUEUES

A message queue is a queue that is generated when an application is deployed. Messages are sent to these queues where they await the message listener.

Report a bug

A.41. MESSAGE LISTENERS

Message listeners encapsulate the communications end-points needed to receive SB-aware messages. Listeners are defined by services and their role is to monitor queues. They receive any messages as they land in those queues. When a listener receives a message, the ESB server calls the appropriate action class defined in the action definition. The methods in this class process the message. In other words, listeners act as inbound routers, directing messages to the action pipeline. When the message has been modified by the actions on the pipeline, the listener sends the result to the replyTo end-point.

You can configure various parameters for listeners. For instance, you can set the number of active worker threads.

There are two types of listeners: ESB-aware listeners and gateway listeners. Gateway listeners are different from ESB-aware listeners in that they accept data in different formats (such as objects in files, SQL tables and JMS messages). They then convert them from these formats to the ESB messaging format. By contrast, ESB-aware listeners can only accept messages that are in the **`org.jboss.soa.esb.message.Message`** format. Each gateway listener must have a corresponding ESB listener defined.

With ESB-aware listeners, RuntimeExceptions can trigger rollbacks. By contrast, with a gateway listener, the transaction simply sends the message to the JBoss ESB. The message is then processed asynchronously. In this way, message failures are separated from message receipts.

Report a bug

A.42. ESB-AWARENESS

If application clients and services are referred to as being ESB-aware, this means that they can understand the message format and transport protocols used by the SOA Platform's enterprise service bus.

Report a bug

A.43. GATEWAY LISTENER

A gateway listener is used to bridge the ESB-aware and ESB-unaware worlds. It is a specialized listener process that is designed to listen to a queue for ESB-unaware messages that have arrived through an external (ESB-unaware) end-point. The gateway listener receives the messages as they land in the queue. When a gateway listener "hears" incoming data arriving, it converts that data (the non-ESB messages) into the **`org.jboss.soa.esb.message.Message`** format. This conversion happens in a variety of different ways, depending on the gateway type. Once the conversion has occurred, the gateway listener routes the data to its correct destination.

Report a bug

A.44. SENDERS

Senders are created by QueueSessions. There is a sender for each queue. The Sender's **`send`** method is called by its QueueSession's ObjectMessage when **`ant runtest`** is executed. When this happens, the client sends a message to the queue.

Report a bug

A.45. JBOSS RULES

JBoss Rules is the name of the business rule engine provided as part of the JBoss Enterprise SOA Platform product.

Report a bug

A.46. RULE BASE

Rule bases are collections of rules. They are used in processing events. The rules help determine how information is stored and processed, which actions are allowed and what action to take when a message is being sent.

Report a bug

A.47. SERIALIZE

To serialize an object is to convert it to a data object.

Report a bug

A.48. DESERIALIZE

To deserialize a file is to transform it back into an object. It is the opposite of serialization.

Report a bug

APPENDIX B. GLOBAL CONFIGURATION FILE

B.1. JBOSSESB-PROPERTIES.XML

The **`jbossesb-properties.xml`** file is the JBoss Enterprise SOA Platform's global configuration file. Many tasks will require you to edit this file. The location of this file varies depending on how the system has been installed. If you have installed a server deployment, this file will be located at **`SOA_ROOT/jboss-as/server/PROFILE/deployers/esb.deployer/jbossesb-properties.xml`**, while standalone clients can access it directly through the class-path.

Report a bug

B.2. GLOBAL CONFIGURATION FILE REFERENCE

The global configuration file (**`SOA_ROOT/jboss-as/server/PROFILE/deployers/esb.deployer/jbossesb-properties.xml`**) is split into sections, each concerned with a specific area of configuration. A named property section contains one or more properties which are used to configure the behavior of the ESB, and one property section can "depend" on another section - the dependency specifies which sections are loaded by the PropertyManager first.

core

- org.jboss.soa.esb.jndi.server.context.factory : The JNDI Server initial context factory.
- org.jboss.soa.esb.jndi.server.url : The JNDI Server URL.
- org.jboss.soa.esb.loadbalancer.policy : The ESB load balancer policy.
- org.jboss.soa.esb.mime.text.types : A semicolon-separated list of MIME types that are used to decide whether the payload can be decoded or whether it will remain as a byte array.
- jboss.esb.invm.scope.default : The default InVM scope for an ESB deployment.
- org.jboss.soa.esb.deployment.schema.validation : A true/false flag to enable JBoss ESB schema validation upon deployment.

IMPORTANT

The org.jboss.soa.esb.jndi.server.type and org.jboss.soa.esb.persistence.connection.factory properties within core are now obsolete.

security

- org.jboss.soa.esb.services.security.implementationClass : The concrete SecurityService implementation to be used.
- org.jboss.soa.esb.services.security.callbackHandler : The default callback handler implementation.
- org.jboss.soa.esb.services.security.sealAlgorithm : The algorithm to be used when sealing the SecurityContext.

- org.jboss.soa.esb.services.security.sealKeySize : The size of the key to be used to encrypt/decrypt the SecurityContext.
- org.jboss.soa.esb.services.security.contextTimeout : The amount of time for which SecurityContext is valid.
- org.jboss.soa.esb.services.security.contextPropagatorImplementationClass : Used to configure a global SecurityContextPropagator.
- org.jboss.soa.esb.services.security.publicKeystore : Keystore to encrypt and decrypt data external to the ESB.
- org.jboss.soa.esb.services.security.publicKeystorePassword : The keystore password.
- org.jboss.soa.esb.services.security.publicKeyAlias : The public key alias to use.
- org.jboss.soa.esb.services.security.publicKeyPassword : The public key password to use.
- org.jboss.soa.esb.services.security.publicKeyTransformation : The cipher transformation to use.

registry

- org.jboss.soa.esb.registry.queryManagerURI : The registry query manager URI, which is used to obtain information on services and bindings.
- org.jboss.soa.esb.registry.lifeCycleManagerURI : The registry lifecycle manager URI, which is used to publish information on services and bindings.
- org.jboss.soa.esb.registry.securityManagerURI : The registry security manager URI, which is used to authenticate queries to the registry.
- org.jboss.soa.esb.registry.implementationClass : The JBoss ESB registry implementation class. The JAXR registry implementation is used here.
- org.jboss.soa.esb.registry.factoryClass : The registry factory class, which specifies which JAXR implementation should be used.
- org.jboss.soa.esb.registry.user : The registry user.
- org.jboss.soa.esb.registry.password : The registry password.
- org.jboss.soa.esb.scout.proxy.transportClass The Scout transport class which defines which transport should be used to communicate with the UDDI registry.
- org.jboss.soa.esb.scout.proxy.uddiVersion : The Scout UDDI Version. This is an Apache Scout-specific setting.
- org.jboss.soa.esb.scout.proxy.uddiNameSpace : The Scout UDDI namespace. This is an Apache Scout-specific setting.
- org.jboss.soa.esb.registry.interceptors : The registry interceptor class names.
- org.jboss.soa.esb.registry.cache.maxSize : The maximum cache size for the caching registry.
- org.jboss.soa.esb.registry.cache.validityPeriod : The validity period for the caching registry.

- org.jboss.soa.esb.registry.orgCategory : The UDDI organization value to use - note that this is a UDDI-specific value.

transports

- org.jboss.soa.esb.mail.smtp.host : The host name of the SMTP server.
- org.jboss.soa.esb.mail.smtp.user : The username to use for the SMTP server.
- org.jboss.soa.esb.mail.smtp.password : The password for the user specified on the SMTP server.
- org.jboss.soa.esb.mail.smtp.port : The port number of the SMTP server.
- org.jboss.soa.esb.mail.smtp.auth : Flag which specifies whether to authenticate the user against the SMTP server using the AUTH command.
- org.jboss.soa.esb.ftp.localdir : FTP local directory.
- org.jboss.soa.esb.ftp.remotedir : FTP remote directory.
- org.jboss.soa.esb.ftp.timeout : FTP timeout in milliseconds for opening a socket.
- org.jboss.soa.esb.ftp.timeout.data : FTP timeout in milliseconds for the data connection.
- org.jboss.soa.esb.ftp.timeout.so : FTP timeout in milliseconds used for currently open sockets.
- org.jboss.soa.esb.ftp.timeout.default : FTP timeout in milliseconds which sets the default timeout.
- org.jboss.soa.esb.jms.connectionPool : Size of the ESB JMS connection pool.
- org.jboss.soa.esb.jms.sessionSleep : If a JMS session cannot be obtained, the ESB will keep trying to obtain one. The sessionSleep property decides how long the ESB will try for.
- org.jboss.soa.esb.invm.expiryTime : The expiry time for messages in the InVM temporary transport.
- org.jboss.soa.esb.invm.retry.limit : Maximum number of times to retry redelivery. The default is 5.
- org.jboss.soa.esb.ws.returnStackTrace : True/false flag that determines whether to return stack traces upon fault of SOAP messages.
- org.jboss.soa.esb.ws.timeout : Service invoker timeout for delivering SOAP messages within RequestResponseBaseWebService.
- org.jboss.soa.esb.aggregator.setOnProperties : Aggregate on properties of the message rather than on Context.

jca

- org.jboss.soa.esb.jca.activation.mapper.jms-ra.rar : Specifies the ActivationMapper globally.
- org.jboss.soa.esb.jca.activation.mapper.wmq.jmsra.rar : Specifies the ActivationMapper globally.

dbstore

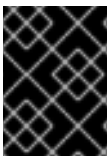
- org.jboss.soa.esb.persistence.db.conn.manager : Connection Manager implementation class name.
- org.jboss.soa.esb.persistence.db.datasource.name : Datasource name, only used if using the J2EE connection manager.
- org.jboss.soa.esb.persistence.db.connection.url : The JDBC connection URL.
- org.jboss.soa.esb.persistence.db.jdbc.driver : The JDBC driver class.
- org.jboss.soa.esb.persistence.db.user : The database user.
- org.jboss.soa.esb.persistence.db.pwd : The database password.
- org.jboss.soa.esb.persistence.db.pool.initial.size : The initial number of database connections.
- org.jboss.soa.esb.persistence.db.min.size : The minimum number of database connections.
- org.jboss.soa.esb.persistence.db.max.size : The maximum number of database connections.
- org.jboss.soa.esb.persistence.db.pool.test.table : A table name to query for validity of the database connection.
- org.jboss.soa.esb.persistence.db.pool.timeout.millis : Timeout in milliseconds of the database connections.

filters

- org.jboss.soa.esb.filter.1, org.jboss.soa.esb.filter.2, org.jboss.soa.esb.filter.3, etc.

rules

- org.jboss.soa.esb.services.rules.resource.scanner.interval : Defines the polling interval for DRL changes globally across all KnowledgeAgents.
- org.jboss.soa.esb.services.rules.continueState : Setting this property will enable legacy behaviour and not dispose of working memories during stateful rule execution.



IMPORTANT

The message routing properties (org.jboss.soa.esb.routing.cbrClass) are obsolete but may still exist in some files.

Report a bug

APPENDIX C. ESB ARCHIVES

C.1. TYPES OF JAVA ARCHIVES

The JBoss Enterprise Application Platform recognizes several different types of archive files. Archive files are used to package deployable services and applications.

In general, archive files are Zip archives, with specific file extensions and specific directory structures. If the Zip archive is extracted before being deployed on the application server, it is referred to as an exploded archive. In that case, the directory name still contains the file extension, and the directory structure requirements still apply.

Table C.1.

Archive Type	Extension	Purpose	Directory structure requirements
Java Archive	.jar	Contains Java class libraries.	**`META-INF/MANIFEST.MF`** file (optional), which specifies information such as which class is the **`main`** class.
Web Archive	.war	Contains Java Server Pages (JSP) files, servlets, and XML files, in addition to Java classes and libraries. The Web Archive's contents are also referred to as a Web Application.	**`WEB-INF/web.xml`** file, which contains information about the structure of the web application. Other files may also be present in **`WEB-INF/`**.
Resource Adapter Archive	.rar	The directory structure is specified by the JCA specification.	Contains a Java Connector Architecture (JCA) resource adapter. Also called a connector.
Enterprise Archive	.ear	Used by Java Enterprise Edition (EE) to package one or more modules into a single archive, so that the modules can be deployed onto the application server simultaneously. Maven and Ant are the most common tools used to build EAR archives.	**`META-INF/`** directory, which contains one or more XML deployment descriptor files.

Archive Type	Extension	Purpose	Directory structure requirements
			Any of the following types of modules. • A Web Archive (WAR). • One or more Java Archives (JARs) containing Plain Old Java Objects (POJOs). • One or more Enterprise JavaBean (EJB) modules, containing its own **`META-INF/`** directory. This directory includes descriptors for the persistent classes which are deployed. • One or more Resource Archives (RARs).
Service Archive	.sar	Similar to an Enterprise Archive, but specific to the JBoss Enterprise Application Platform.	**`META-INF/`** directory containing **`jboss-service.xml`** or **`jboss-beans.xml`** file.

Report a bug

C.2. ESB ARCHIVE

An ESB archive is a special type of JAR file. It contains all of the files that make up a deployable ESB project.

Report a bug

C.3. DEPLOY AN ARCHIVE

Procedure C.1. Task

- To deploy an archive to your server, copy it to the **`deploy`** directory: **`cp FILENAME.esb SOA_ROOT/jboss-as/server/PROFILE/deploy`**.

Result

The directory is being polled by the server, so it will find the archive immediately. Note you can also deploy *.war files in archived or uncompressed form.

Report a bug

C.4. STRUCTURE OF AN ESB ARCHIVE

An ESB archive file consists of a number of other files, as follows:

At the archive's root level, there are these files:

***-ds.xml (for example, message-store-ds.xml or quickstart-ds.xml)**

These are database scripts.

***-service.xml (for example, jbm-queue-service.xml)**

Services, including the Admin objects for queues, and one that initializes the database using above script

hsqldb

For a database example. Below it resides a **`create.sql`** file that makes the database.

Under the archive's **`META_INF`** directory, there are these files:

deployment.xml

This lists the dependencies required by the .esb

jboss-esb.xml

This is the deployment descriptor for this .esb

MANIFEST.MF

The manifest file.

Finally, nested under the org/jboss/soa/esb directories there are custom actions and database scripts.

Report a bug

APPENDIX D. REVISION HISTORY

Revision 5.3.1-69.400 **2013-10-31** **Rüdiger Landmann**

Rebuild with publican 4.0.0

Revision 5.3.1-69 **Tue Feb 05 2013** **David Le Sage**

Built from Content Specification: 6585, Revision: 371683 by dlesage