JBoss Enterprise SOA Platform 5

ESB Services Guide

This guide is for developers.

Edition 5.3.1

Last Updated: 2017-10-27

JBoss Enterprise SOA Platform 5 ESB Services Guide

This guide is for developers.
Edition 5.3.1

David Le Sage
Red Hat Engineering Content Services
dlesage@redhat.com

B Long
Red Hat Engineering Content Services
belong@redhat.com

Legal Notice

Abstract

This guide teaches developers how to develop services for the JBoss Enterprise SOA Platform.

Table of Contents

PREFACE

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?

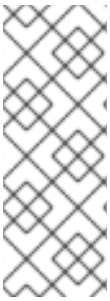
Introduction

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

PART I. INTRODUCTION

CHAPTER 2. INTRODUCTION

2.1. INTENDED AUDIENCE

This book has been designed to be understood by developers wanting to learn the basics about developing services for the JBoss Enterprise SOA Platform.

Report a bug

2.2. AIM OF THIS BOOK

Aim

The Enterprise Service Bus Services Guide aims to teach developers how to create services for deployment to the JBoss Enterprise SOA Platform. Readers will learn how to use web applications, configure rule services and the content-based routing feature, transform messages and deploy services.

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

CHAPTER 3. BASICS

3.1. OUT-OF-THE-BOX ACTIONS

Out-of-the-box actions are generic pieces of code for actions that come prepackaged with the JBoss Enterprise SOA Platform product. You can use them immediately in your services or customize them to suit your needs.

Report a bug

3.2. JBOSS ENTERPRISE SOA PLATFORM OUT-OF-THE-BOX ACTIONS

The out-of-the-box actions implemented in the SOA Platform are divided into the following functional groups:

Transformers and Converters

Use transformer and converter actions to change message data from one form to another.

Business Process Management

Use the business process management actions when integrating your software with the jBPM.

Scripting

Use scripting actions to automate tasks written in the supported scripting languages.

Services

Use service actions when integrating your code with Enterprise Java Beans.

Routing

Use routing actions when moving message data to destination services.

Notifier

Use notifier actions when sending data to ESB-unaware destinations.

Web Services/SOAP

Use web service actions when you need to support web services.

Report a bug

3.3. 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

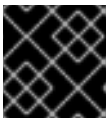
3.4. IMPORTANT NOTES ABOUT QUICKSTARTS

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

3.5. LEARN MORE ABOUT A QUICKSTART

To learn more about a particular quickstart:

Procedure 3.1. Task

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

Report a bug

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

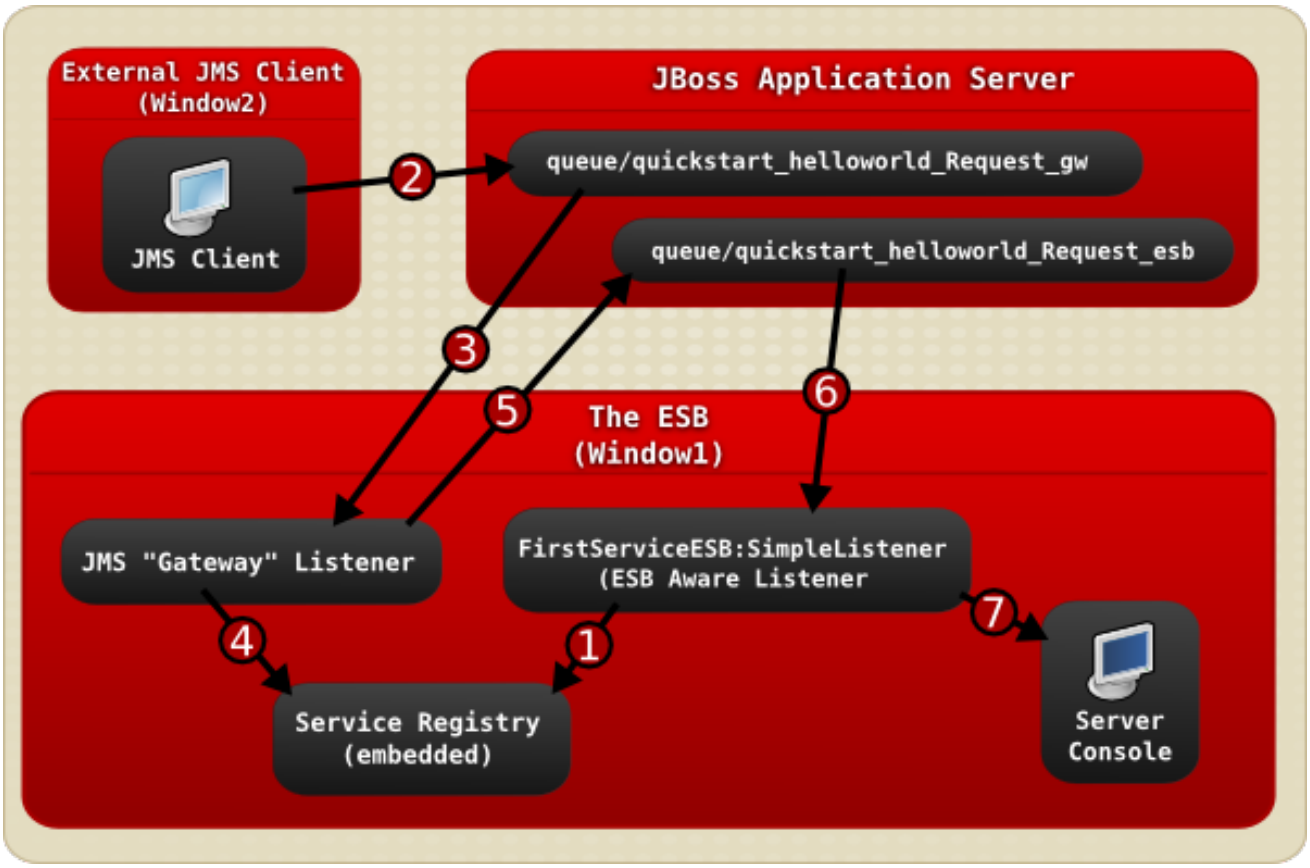


Figure 3.1. Image

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

PART II. SERVICE REGISTRATION AND HOSTING

CHAPTER 4. INTRODUCING THE SERVICE REGISTRY

4.1. ABOUT THIS SECTION

Introduction

Read this section to learn what a service registry is and how it interacts with the ESB component. To learn how to develop for the Registry, read the *jUDDI Registry Guide*.

Report a bug

4.2. 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

4.3. JUDDI REGISTRY

The jUDDI (Java Universal Description, Discovery and Integration) Registry is a core component of the JBoss Enterprise SOA Platform. It is the product's default service registry and comes included as part of the product. In it are stored the addresses (end-point references) of all the services connected to the Enterprise Service Bus. It was implemented in JAXR and conforms to the UDDI specifications.

Report a bug

4.4. 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

4.5. OTHER SUPPORTED SERVICE REGISTRIES

The JBoss Enterprise SOA Platform also supports these other UDDI registries:

- SOA Software SMS
- HP Systinet

Report a bug

4.6. SERVICE PROVIDER

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

Report a bug

4.7. 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

4.8. 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

4.9. 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

4.10. 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

4.11. 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

4.12. UNIVERSAL DESCRIPTION, DISCOVERY AND INTEGRATION (UDDI) REGISTRY

The Universal Description, Discovery and Integration Registry (UDDI) is a directory for web services. Use it to locate services by running queries through it at design- or run-time. Within an UDDI Registry, information is categorized in Pages. UDDI creates a standard interoperable platform that enables companies and applications to quickly, easily, and dynamically find and use Web services over the Internet. UDDI also allows operational registries to be maintained for different purposes in different contexts.

The UDDI also allows providers to publish descriptions of their services. The typical UDDI Registry will contain a uniform resource locator (URL) that points to both the WSDL document for the web services and the contact information for the service provider.

A business publishes services to the UDDI registry. A client looks up the service in the registry and receives service binding information. The client then uses the binding information to invoke the service. The UDDI APIs are SOAP-based for interoperability reasons.

Report a bug

4.13. UDDI APPLICATION PROGRAMMING INTERFACES

The UDDI v3 specification defines nine APIs:

UDDI_Security_PortType

This defines the API to obtain a security token. With a valid security token a publisher can publish to the registry. A security token can be used for the entire session.

UDDI_Publication_PortType

This defines the API to publish business and service information to the UDDI registry.

UDDI_Inquiry_PortType

This defines the API to query the UDDI registry. This API does not normally require a security token.

UDDI_CustodyTransfer_PortType

This API can be used to transfer the custody of a business from one UDDI node to another.

UDDI_Subscription_PortType

This defines the API to register for updates on a particular business of service.

UDDI_SubscriptionListener_PortType

This defines the API a client must implement to receive subscription notifications from a UDDI node.

UDDI_Replication_PortType

This defines the API to replicate registry data between UDDI nodes.

UDDI_ValueSetValidation_PortType

This is used by nodes to allow external providers of value set the validation. Web services to assess whether keyedReferences or keyedReferenceGroups are valid.

UDDI_ValueSetCaching_PortType

UDDI nodes may perform validation of publisher references themselves using the cached values obtained from such a Web service.

Report a bug

4.14. UDDI PAGE TYPES

Green Pages

Green Pages provide information that enables you to bind a client to the service being provided.

Yellow Pages

Yellow Pages are used to categorize businesses based upon the industries to which they belong.

White Pages

White Pages contain general information, such as the name, address and other contact details for the company providing the service.

Report a bug

4.15. THE SERVICE REGISTRY AND THE JBOSS ENTERPRISE SOA PLATFORM

The Service Registry is a key part of the JBoss Enterprise SOA Platform. When you deploy services to SOA Platform's ESB, their end-point references are stored in it.

Report a bug

4.16. JUDDI AND THE ESB

The JBoss Enterprise Service Bus directs all interaction with the Registry through the registry interface, the default version of which uses Apache Scout.

Report a bug

4.17. 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

CHAPTER 5. PUBLISHING CONTRACTS

5.1. 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

5.2. END-POINT CONTRACT

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

Report a bug

5.3. 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

5.4. PUBLISH A CONTRACT

Procedure 5.1. 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

PART III. SERVICE ORCHESTRATION AND BUSINESS PROCESS MANAGEMENT

CHAPTER 6. JBPM WEB APPLICATIONS

6.1. 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

6.2. JBPM AND ESB INTEGRATION

The JBoss ESB is integrated with the jBPM component for two reasons:

- Service Orchestration: You can orchestrate services using the Business Process Manager by creating a process definition.
- Human Task Management: The Business Process Manager allows you to integrate machine-based services with the management of tasks undertaken by people.

Report a bug

6.3. CREATE A GRAPHICAL REPRESENTATION OF THE STEPS IN A BUSINESS PROCEDURE

Procedure 6.1. Task

- Use jBPM's Process Designer functionality.

NOTE

A side benefit of using this tool is that it can help foster good working relationships between your business analysts and your technical developers.

Report a bug

6.4. 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

6.5. DEPLOYING A JBPM WEB APPLICATION TO THE JBOSS ENTERPRISE SOA PLATFORM

1. You have many deployment options: choose from using the **`GPD deployment`** (Graphical Design Process) tab, the **`JSF console upload form`** and the **`Ant DeployProcessTask`**.

WARNING

Do not include the jBPM libraries in your deployed ESB applications. The **`jbpm.esb`** module already provides the libraries and configuration files required to run jBPM applications. Always use the provided versions and default settings as these have been refined by Red Hat's extensive Quality Engineering tests in order to prevent issues such as class loading conflicts and configuration mismatches.

NOTE

Process definitions should be deployed separately from the web application. Red Hat recommends that you deploy the process before the web application so that the latter can operate under the assumption the process is available at all times.

2. The jBPM Graphical Design Process editor includes four modes: Diagram, Deployment, Design and Source, which are available as switchable tabs at the bottom of the editor. To adjust the deployment settings of the project you should select the tab that opens the Deployment mode. You can easily modify them or, if the settings do not match your needs, reset them to their defaults.

3. In multi-tenancy use cases, a single server hosts many applications, each of which requires a different configuration. Red Hat recommends that you give each configuration file a unique name (something other than jbpm.cfg.xml), to avoid overriding the default configuration file that comes provided with the platform.

Report a bug

CHAPTER 7. JBPM 3 INTEGRATION

7.1. JBOSS BUSINESS PROCESS MANAGER

The JBoss Business Process Manager (jBPM) is a work-flow and business process management engine. It allows you to co-ordinate people, applications and services as you design your business processes.

Report a bug

7.2. JBPM INTEGRATION CONFIGURATION

1. To create a JBPM database, start the **`DatabaseInitializer`***MBean*. (You will find the configuration settings for this MBean in the first configuration element of the **`SOA_ROOT/jboss-as/server/PROFILE/deploy/jbpm.esb/jbpm-service.xml`** file.)

WARNING

The JbpmDS data source is defined in the jbpm-ds.xml file, located in **`SOA_ROOT/jboss-as/server/PROFILE/deploy/jbpm.esb`**. By default, it uses a Hypersonic database. Always change this to a production-quality database in a live environment.

WARNING

The JBoss Enterprise SOA Platform also comes equipped with Hypersonic, an in-memory reference database. Use this in testing environments only.

2. Follow the example shown below:

```
<classpath codebase="deploy" archives="jbpm.esb"/>
<classpath codebase="deploy/jbossesb.sar/lib"
   archives="jbossesb-rosetta.jar"/>

<mbean
code="org.jboss.internal.soa.esb.dependencies.DatabaseInitializer"
       name="jboss.esb:service=JBPMDatabaseInitializer">
      <attribute name="Datasource">java:/JbpmDS</attribute>
      <attribute name="ExistsSql">select count(*) from
JBPM_ID_USER</attribute>
      <attribute name="SqlFiles">
      jbpm-sql/jbpm.jpdl.hsqldb.sql
```

```
        </attribute>

<depends>jboss.jca:service=DataSourceBinding,name=JbpmDS</depends>
        <attribute name="UseEOL">true</attribute>
</mbean>

<mbean
code="org.jboss.soa.esb.services.jbpm.configuration.JbpmService"
   name="jboss.esb:service=JbpmService">
</mbean>
```

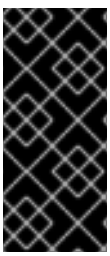
Report a bug

7.3. JBPM 5 TO JBOSS ESB INTEGRATION

jBPM 5-to-JBossESB communication lets you use the JBPM for service orchestration. The two JBPM work item handler classes used to integrate these services are the EsbActionWorkItemHandler and the ESBServiceWorkItemHandler. The EsbActionWorkItemHandler is a request-reply type action that sends a message to a service and waits for a response. By contrast, the EsbServiceWorkItemHandler does not wait for a response.

You must provide a callback service within their jboss-esb.xml which contains the BPM5Callback action. The callback service's category and name are provided to the ESBActionWorkItemHandler so that it can contact the callback service. The following is an example of the configuration:

```
<service category="EsbJbpm5Example"
                 name="JBpm5CallbackService"
                 description="Service which makes Callbacks into jBPM">
            <listeners>
                <jms-listener name="JMS-DCQListener"
                              busidref="jBPMCallbackBus"
                              maxThreads="1"
                />
            </listeners>
            <actions mep="OneWay">
                <action name="action"
class="org.jboss.soa.esb.services.jbpm5.actions.Bpm5Callback">
                    <property name="process-definition-name"
value="sample.bpmn"/>
                </action>
            </actions>
        </service>
      </services>
```



IMPORTANT

The process definition names need to be unique between applications. You can define these names in the property shown below:

```
<property name="process-definition-name" value="sample.bpmn"/>
```

Report a bug

7.4. THE DATABASEINITIALIZER MBEAN'S DEFAULT VALUES

Table 7.1. The DatabaseInitializer MBean's Default Values

Property	Description	Default
Datasource	The datasource for the JBPM database.	java:/JbpmDS
ExistsSql	Use this SQL command to confirm the existence of the database.	Select count(*) from JBPM_ID_USER
SqlFiles	These files contain the SQL commands to create the JBPM database if it is not found.	jbpm-sql/jbpm.jpdl.hsqldb.sql, jbpm-sql/import.sql

NOTE

By default, the **DatabaseInitializer** MBean is configured to wait until for the **JbpmDS** is deployed, before it then deploys itself.

Report a bug

7.5. THE JBPMSERVICE MBEAN

The **JbpmService** bean ties the life-cycle of the **JBoss Business Process Manager**'s *job executor* to that of the **jbpm.esb**. It does so by launching a **job executor** instance on start-up and closes it on shutdown.

Report a bug

7.6. CONFIGURING THE JBPM

The configuration settings for the JBoss Business Process Manager are stored in three files within the **SOA_ROOT/jboss-as/server/PROFILE/deploy/jbpm.esb/** directory:

- **jbpm.cfg.xml**
- **hibernate.cfg.xml**
- **jbpm.mail.templates.xml**

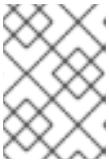
1. The **jbpm.cfg.xml** file is set to tell the JBPM to use the JTA Transaction Manager:

```
<service name="persistence">
      <factory>
        <bean
class="org.jbpm.persistence.jta.JtaDbPersistenceServiceFactory">
          <property name="isTransactionEnabled"><false/></property>
```

```
          <property
name="isCurrentSessionEnabled"><true/></property>
          <!--property name="sessionFactoryJndiName">
            <string value="java:/myHibSessFactJndiName" />
          </property-->
        </bean>
     </factory>
     </service>
```

2. The **hibernate.cfg.xml** file also tells the JBPM use the **JTA Transaction Manager**:

```
    <!-- JTA transaction properties (begin) -->
    <property name="jta.UserTransaction">UserTransaction</property>
    <property
name="hibernate.current_session_context_class">jta</property>
    <property
name="hibernate.transaction.factory_class">org.hibernate.transaction
.JTATransactionFactory</property>
    <property
name="hibernate.transaction.manager_lookup_class">org.hibernate.tran
saction.JBossTransactionManagerLookup</property>
    <!-- JTA transaction properties (end) -->
```



NOTE

Do not use **Hibernate** to create the database schema. Use the **DatabaseInitializer** MBean instead.

3. The **jbpm.mail.templates.xml** file contains the following:

```
        jboss-as ]$ cat
server/default/deploy/jbpm.esb/jbpm.mail.templates.xml

<?xml version="1.0" encoding="UTF-8"?>
<mail-templates>
  <variable name="taskListBaseURL"
value="http://localhost:8080/jbpm-console/app/task.jsf?id=" />

  <mail-template name='task-assign'>
    <actors>${taskInstance.actorId}</actors>
    <subject>Task notification: ${taskInstance.name}</subject>
    <text><![CDATA[Hi ${taskInstance.actorId},
Task '${taskInstance.name}' has been assigned to you.
Go for it: ${taskListBaseURL}${taskInstance.id}

Sent by jBPM]]></text>
  </mail-template>

  <mail-template name='task-reminder'>
    <actors>${taskInstance.actorId}</actors>
    <subject>Task reminder: ${taskInstance.name}</subject>
    <text><![CDATA[Hey ${taskInstance.actorId},
Do not forget about task '${taskInstance.name}'.
```

```
Get going: ${taskListBaseURL}${taskInstance.id}

Sent by jBPM]]></text>
  </mail-template>
</mail-templates>
```

NOTE

To learn more about each of these configuration files, refer to the *JBPM Reference Guide*.

Report a bug

7.7. CREATING A PROCESS DEFINITION

1. Use a Creation wizard to create an empty process definition. Select **File** → **New** → **Other**. The wizard opens on the *Select Wizard* page.
2. Select the *JBoss jBPM* category, then the *jBPM Process Definition* item. Clicking on the **Next** button brings us to the *Create Process Definition* page.
3. Type in a name for the process archive file. Click on the **Finish** button to end the wizard and open the process definition editor.
4. By viewing the Package Explorer, you can see that creating a process definition involves creating an XML file called **`[process name].jpdl.xml`**, which contains the process definition information. A JPG file called **`[process name].jpg`** will also be automatically generated when changes are saved to the process.

Report a bug

7.8. DEPLOYING A PROCESS DEFINITION

1. Check that the server is running.
2. Activate a *process archive* by going to the **JBPM Graphical Editor**'s **`Deployment`** tab.

NOTE

Sometimes you will only need to deploy the **`processdefinition.xml`** file but, most often, you will be deploying other types of **`artifacts`** as well, such as *task forms*.

3. Deploy the definition using one of these methods:
    1. Using **JBoss Developer Studio** , configure the upload servlet used by the deployer. Next, click the **`Deploy Process Archive`** button. This is visible in the **`Deployment`** view.
    2. Using the **DeployProcessToServer** JBPM **`ant`** task.

3. Save the deployment to a local **.par** file from the **Deployment** view. Next, activate the archive using the JBPM console. (This requires that you have administration privileges.)

Report a bug

7.9. JBPM COMMANDS

The **JBoss ESB** can make calls into the **JBoss Business Process Manager** by means of the **BpmProcessor** action. This action utilizes the **JBPM Command API**. Here are the JBPM commands you can use:

Table 7.2. JBPM Commands

Command	Description
NewProcessInstanceCommand	This command starts a new **ProcessInstance** which is associated with a process definition that has been already deployed to the JBPM. The **NewProcessInstanceCommand** leaves the process instance in the **start** state. This is needed in the case of a task being associated with the **Start node** (such as when there is one on an *actor*'s task-list).
StartProcessInstanceCommand	This is identical to the **NewProcessInstanceCommand** except that the new process instance is automatically moved from the **start** position to the first node.
GetProcessInstanceVariablesCommand	Displays the root node variables for a process instance by using the process instance identifier.
CancelProcessInstanceCommand	Cancels an entire **ProcessInstance**. (Requires some JBPM context variables to be set on the message, including the **ProcessInstance** identifier.)

Report a bug

7.10. CONFIGURING A NEW PROCESS INSTANCE IN JBPM

1. The configuration for this action in the jboss-exb.xml looks like this:

```
<action name="create_new_process_instance"
class="org.jboss.soa.esb.services.jbpm.actions.BpmProcessor">
 <property name="command" value="StartProcessInstanceCommand" />
 <property name="process-definition-name"
value="processDefinition2"/>
 <property name="actor" value="FrankSinatra"/>
 <property name="esbToBpmVars">
  <!-- esb-name maps to getBody().get("eVar1") -->
```

```
    <mapping esb="eVar1" bpm="counter" default="45" />
    <mapping esb="BODY_CONTENT" bpm="theBody" />
   </property>
  </action>
```

2. You are required to input these two attributes:

    1. name

       Use any value for this name attribute, as long as it is unique in the **`action pipeline`**.

    2. class

       Always set this attribute to **`org.jboss.soa.esb.services.jbpm.actions.BpmProcessor`**.

Report a bug

7.11. JBPM CONFIGURATION PROPERTIES

Table 7.3. JBPM Configuration Properties

Property	Description	Required?
command	This must be one of: **`NewProcessInstanceCommand`**, **`StartProcessInstanceCommand`**, **`GetProcessInstanceVariablesCommand`** or **`CancelProcessInstanceCommand`**.	Yes
process-definition-name	Required for the **`NewProcessInstanceCommand`** and **`StartProcessInstanceCommand`** if the process-definition-id property is not used. The value of this property should reference the already-deployed process definition that needs a new instance. (This property does not apply to the **`CancelProcessInstanceCommand`**.)	Sometimes
process-definition-id	Required for the **`NewProcessInstanceCommand`** and **`StartProcessInstanceCommand`** if the process-definition-name property is not used. The value of this property should refer to the already-deployed process definition for which a new instance is to be created. (This property does not apply to the **`CancelProcessInstanceCommand`**.)	Sometimes
actor	Specifies the **`JBPM actor`** identifier. (Only applies to the **`NewProcessInstanceCommand`** and the **`StartProcessInstanceCommand`** .)	No

Property	Description	Required?
key	Specifies the value of the JBPM key. The key is a string-based business key property on the process instance. The combination of business key and process definition must be unique if a business key is supplied. The key value can hold an MVEL expression to extract the desired value from the EsbMessage. For example, if you were to have a named parameter called **`businessKey`** in the body of a message, body.businessKey would be used. (This property only applies to **`NewProcessInstanceCommand`** and **`StartProcessInstanceCommand`**.)	No
transition-name	This only applies to **`StartProcessInstanceCommand`**. Use it only if there is more than one transition out of the current node. If this property is not specified, then the default transition out of the node is taken. The default transition is the first transition in the list of transitions defined for that node in the JBPM **`processdefinition.xml`**.	No

Property	Description	Required?
esbToBpmVars	This is an optional property for the **`New-`** and **`StartProcessInstanceCommand`**. It defines a list of variables which need to be extracted from the ESB Message and set into the JBPM context for that particular process instance. The list consists of mapping elements, each of which can have the following attributes: • esb This is a required attribute. Place an MVEL expression in it and use it to extract a value from anywhere in the ESB message. • bpm This is an optional attribute containing the name to use on the JBPM side. (If it is omitted, the Enterprise Service Bus name is used instead.) • default This is an optional attribute which can hold a default value if the ESB's MVEL expression does not find a value set in the ESB message. • bpmToEsbVars This is structurally identical to the esbToBpmVars property above. Use it in conjunction with the **`GetProcessInstanceVariablesCommand`** to map JBPM **`process instance variables`** (**`root token`** variables) to the ESB message. • reply-to-originator This is an optional property for the **`New`** - and **`StartProcessInstanceCommand`**. Specify a value of **`true`** to make the process instance store the **`ReplyTo`** / **`FaultTo`** values of the invoking message's endpoint references within the process instance. These values can then be used within subsequent **`EsbNotifier`** / **`EsbActionHandler`** invocations to deliver a message to the **`ReplyTo`** / **`FaultTo`** addresses.	No

Report a bug

7.12. ESBMESSAGE BODY CONFIGURATION IN JBPM

Table 7.4. EsbMessage Body Configuration in JBPM

Property	Description
jbpmProcessInstId	This is a required ESB message Body parameter that applies to the **`GetProcessInstanceVariablesCommand`** and **`CancelProcessInstanceCommand`** commands. Manually set this as a named parameter on the EsbMessage body.

Report a bug

7.13. ESB-TO-JBPM EXCEPTION HANDLING

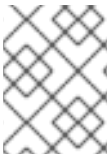
If a **`JbpmException`** is thrown from the JBPM Command API during an ESB call, it is passed to the **`action pipeline`**. The pipeline logs the error, directs the message to the **`DeadLetterService`** and sends an error to the **`faultTo`** endpoint reference (provided it has been set).

Report a bug

7.14. JBPM-JBOSSESB-TO-ESB INTEGRATION

JBPM-JBossESB-to-JBossESB communication lets you use the JBPM for service orchestration.

The two JBPM action handler classes used to intergrate these services are the **`EsbActionHandler`** and **`EsbNotifier`**. The **`EsbActionHandler`** is a request-reply type action that sends a message to a service and waits for a response. By contrast, the **`EsbNotifier`** does not wait for a response.



NOTE

The interaction with the Enterprise Service Bus is *asynchronous* in nature and, therefore, does not block the process instance whilst the service executes.

WARNING

It is important that the classes representing any of the variables passing between JBPM and JBoss ESB services are visible to the JBPM process, target ESB services and the ESB JBPM callback service. Always deploy these classes to the server's **`lib`** directory.

Report a bug

7.15. ESB NOTIFIER ACTION IN JBPM

1. Attach the **EsbNotifier** to the JBPM**processdefinition.xml** file's *outgoing transition* as shown below:

```
<node name="ShipIt">
  <transition name="ProcessingComplete" to="end">
    <action name="ShipItAction"

class="org.jboss.soa.esb.services.jbpm.actionhandlers.EsbNotifier">
      <esbCategoryName>BPM_Orchestration4</esbCategoryName>
      <esbServiceName>ShippingService</esbServiceName>
      <bpmToEsbVars>
        <mapping bpm="entireCustomerAsObject" esb="customer" />
        <mapping bpm="entireOrderAsObject" esb="orderHeader" />
        <mapping bpm="entireOrderAsXML" esb="entireOrderAsXML" />
      </bpmToEsbVars>
    </action>
  </transition>
</node>
```

2. You can specify the following attributes:

   - name

     This is required. It is the user-specified name of the action

   - class

     This is required. You must set it to **org.jboss.soa.esb.services.jbpm.actionhandlers.EsbNotifier**

Report a bug

7.16. CONFIGURING THE ESB ACTION HANDLER

1. Attach the **EsbActionHandler** to a node to call an action when that node is entered. When the **EsbActionHandler** is executed, the node waits for a transition signal (normally sent by the **JBossESB callback** service).

2. Configure as follows:

```
<action name="create_new_process_instance"
  class="org.jboss.soa.esb.services.jbpm.actions.BpmProcessor">

  <property name="command" value="StartProcessInstanceCommand" />
  <property name="process-definition-name"
value="processDefinition2"/>
  <property name="actor" value="FrankSinatra"/>

  <property name="esbToBpmVars">
  <!-- esb-name maps to getBody().get("eVar1") -->
    <mapping esb="eVar1" bpm="counter" default="45" />
    <mapping esb="BODY_CONTENT" bpm="theBody" />
  </property>
```

```
</action>
```

Report a bug

7.17. ESBACTIONHANDLER EXTENSION CONFIGURATION

The **EsbActionHandler** relies on the configuration settings for the **EsbNotifier**. The extensions consist of the following sub-elements:

Table 7.5. EsbActionHandler Extension Configuration

Property	Description	Required?
esbToBpmVars	Identical to the **esbToBpmVars** property for the **BpmProcessor** configuration. This sub-element defines a list of variables that need to be extracted from the ESB message and set in the Business Process Manager context for that particular process instance. If left unspecified, the **globalProcessScope** value defaults to **true** when the variables are set. The list consists of mapping elements, each of which can have the following attributes: • **esb** This is a required attribute which can contain an MVEL expression. Use it to extract a value and put it into the ESB Message from anywhere. • **bpm** This is an optional attribute containing the name which is to be used by the JBPM. If it is not supplied, then the name in **esb** is used instead. • default Use this is an optional attribute to hold a default value if the **esb** MVEL expression cannot find one that is set in the Enterprise Service Bus message. • **process-scope** This is an optional parameter consisting of a Boolean value. Use it to override the setting of this mapping's e **globalProcessScope**.	No
exceptionTransition	This the name of the transition to utilize if an exception occurs while the service is being processed. It requires the current node to have several outgoing transitions, one of which can handle *exception processing*.	No

Report a bug

7.18. PASSING PARAMETERS TO A JBPM5 PROCESS ON STARTPROCESS

This is the code for passing parameters to the jBPM5 process using startProcess. All that is required is to add required parameter as properties to the ESBMessage.

```
// create the ESB message
Message esbMessage = MessageFactory.getInstance().getMessage();
// add a parameter
esbMessage.getProperties().setProperty("name", "Laurel");
```

Report a bug

7.19. PASSING PARAMETERS TO A JBPM5 PROCESS ON SIGNALEVENT

```
// create the ESB message
Message esbMessage = MessageFactory.getInstance().getMessage();
// set the process event type as defined in the process definition
esbMessage.getProperties().setProperty("processEventType", "NewMessage");
// add a parameter
esbMessage.getProperties().setProperty("name", "Hardy");

// setup data required to identify the intended target process instance
ContextImpl ctxi = (ContextImpl) esbMessage.getContext();
// set the session id
ctxi.setContext("jbpm5-session-id", sessionId);
// set the instance id.
ctxi.setContext("jbpm5-processinstance-id", processInstanceId);
```

The following 'name' property will be assigned on process start when the 'name' ESB message property will be set:

```
<definition ...>

    <itemDefinition id="_nameItem" structureRef="String" />

    <process name="Hello" tns:packageName="defaultPackage" ...>
        <property id="name" itemSubjectRef="_nameItem"/>
        <!-- ... -->
    </process>
    <!-- ... -->
</definition>
```

To define a process property of type java.util.Map:

```
<definition ...>

  <itemDefinition id="_objectMapItem" structureRef="java.util.Map" />
```

```
  <process name="Hello" tns:packageName="defaultPackage" ...>

    <property id="objectMap" itemSubjectRef="_objectMapItem"/>
    <!-- ... -->
  </process>
  <!-- ... -->
</definition>
```

To extract parameters:

```
String name2 = objectMap.get("name"); // will retrieve the Hardy string
```

Report a bug

7.20. SIGNAL EVENT EXAMPLE

This is an example of a signal event. It defines data association which will map the map from the ESB message to a variable in a process.

```
<intermediateCatchEvent id="_4" name="Signal" >
      <dataOutput id="_4_Output" name="event" />
      <dataOutputAssociation>
      <sourceRef>_4_Output</sourceRef>
      <targetRef>objectMap</targetRef>
      </dataOutputAssociation>
      <outputSet>
        <dataOutputRefs>_4_Output</dataOutputRefs>
      </outputSet>
      <signalEventDefinition signalRef="NewMessage"/>
</intermediateCatchEvent>
```

Report a bug

7.21. LIST OF ESB NOTIFIER SUB-ELEMENTS

Table 7.6. List of ESB Notifier Sub-Elements

Sub-element	Description
esbCategoryName	This is the ESB service's category name and is required if you are not using the reply-to-originator functionality.
esbServiceName	This is the name of the ESB service and required if you are not using the reply-to-originator functionality.
replyToOriginator	Use this to specify the 'reply' or 'fault' originator address previously stored in the process instance on creation.

Sub-element	Description
globalProcessScope	This element is an optional Boolean-valued parameter. Use it to set the default scope within which the **bpmToEsbVars** variables are to be found. If the **globalProcessScope** is set to **true**, it searches for the variables within the*token hierarchy* (the **process-instance** scope). If it is set to **false**, it retrieves the variables in the scope of the token. If the token itself does not possess a variable for a given name, the token hierarchy is used to search for that variable. If the element is omitted altogether, the **globalProcessScope** defaults to **false**.
bpmToEsbVars	This element is optional. It takes a list of sub-elements and uses them to map a JBPM context variable to an ESB message location. Each of these mapping sub-elements can have the following attributes:
bpm	This is a required attribute. It is the name of the variable in JBPM context. The name can be MVEL type expression so you can extract a specific field from a larger object. The MVEL root is set to the JBPM “ContextInstance”, so for example you can use mapping like: `<mapping bpm="token.name" esb="TokenName" /> <mapping bpm="node.name" esb="NodeName" /> <mapping bpm="node.id" esb="esbNodeId" /> <mapping bpm="node.leavingTransitions[0].name" esb="transName" /> <mapping bpm="processInstance.id" esb="piId" /> <mapping bpm="processInstance.version" esb="piVersion" />` The JBPM context-variable names can also be referenced directly.
esb	Optional. This is the name of the variable in the Enterprise Service Bus Message. It can be an MVEL-type expression. (The attribute value TokenName in the example above is equal to **body.TokenName**. A special value called **BODY_CONTENT** "addresses" the body directly.) By default, the variable is set as a named parameter on the body of the ESB Message. To omit the **esb** attribute, replace it with the value of the **bpm** attribute.

Sub-element	Description
`process-scope`	This attribute is optional. It is a parameter that can contain a Boolean value used to override the setting of the `globalProcessScope` for this mapping.

IMPORTANT

Always activate **debug**-level logging when working on the variable mapping configuration.

Report a bug

7.22. LIST OF ESBSERVICEWORKITEMHANDLER SUB-ELEMENTS

Table 7.7. List of ESBServiceWorkItemHandler Sub-Elements

Name	Description
ServiceCategory	This is the ESB service category name for the service that jBPM 5 will deliver a message to.
ServiceName	This is the ESB service name for the service that jBPM 5 will deliver a message to.

Report a bug

7.23. LIST OF ESBACTIONWORKITEMHANDLER SUB-ELEMENTS

Table 7.8. List of ESBActionWorkItemHandler Sub-Elements

Name	Description
ServiceCategory	This is the ESB service category name for the service that jBPM 5 will deliver a message to.
ServiceName	This is the ESB service name for the service that jBPM 5 will deliver a message to.
CallbackServiceCategory	The service category of the callback service. The callback service must be provided in the jboss-esb.xml.
CallbackServiceName	The service name of the callback service. The callback service must be provided in the jboss-esb.xml.

Name	Description
replyToOriginator	Use this to specify the 'reply' or 'fault' originator address previously stored in the process instance on creation.
jbpm5-session-id	The jbpm 5 session ID of the session which started this process. This is needed so that the callback service can complete the current work item.

Report a bug

7.24. ADDING A TIME-OUT VALUE IN JBPM

- Add a JBPM-native *timer* to the appropriate node. In this example, the timer has been configured so a transition called **`time-out`** is triggered if there is no signal received in ten seconds:

```
<timer name='timeout' duedate='10 seconds' transition='time-out'/>
```

Report a bug

7.25. JBPM-TO-ESB EXCEPTION HANDLING

Table 7.9. JBPM-to-ESB Exception Handling

Error	Resolution
Delivery error	Add an **`exceptionhandler`** (**`TB-JBPM-USER`**) to the JBPM node to deal with **`MessageDeliveryException`** caused by the user misspelling the service's name. (See http://docs.jboss.com/jbpm/v3/userguide/processmodelling.html for more information.)
Processing error	Sometimes the service receives a request but throws an error during processing. If the call is made from the **`EsbActionHandler`**, the exception reported back to **JBoss Business Process Manager**.

Report a bug

7.26. EXCEPTION HANDLING EXAMPLES

`Time-out`: If you are using the **`EsbActionHandler`** action and the node is awaiting a callback, you can limit the waiting period. To do so, add a timer to the node. (That is how **`Service1`** is configured in

the process definition snippet below.) The timer can be set for a certain period, in this case, ten seconds:

```
<node name="Service1">

  <action class=
    "org.jboss.soa.esb.services.jbpm.actionhandlers.EsbActionHandler">
    <esbCategoryName>MockCategory</esbCategoryName>
      <esbServiceName>MockService</esbServiceName>
  </action>

  <timer name='timeout' duedate='10 seconds'
    transition='time-out-transition'/>
  <transition name="ok" to="Service2"></transition>
  <transition name="time-out-transition" to="ExceptionHandling"/>

</node>
```

Service1 has two outgoing transitions. The first of these is **ok** whilst the second one is **time-out-transition**.

Normally the call-back will signal the default transition, which is **ok** , since it is defined as the first. However, if the processing of the service takes more then ten seconds, the timer will run instead. The timer's transition attribute is set to **time-out-transition**, meaning that this transition will be taken on timing-out.

The processing ends up in the **ExceptionHandling** node. From here, you can perform compensatory work.

Exception Transition: You can define an **exceptionTransition** to handle any exceptions that occurs in the midst of the service being processed. Doing so sets the **faultTo** endpoint reference on the message, meaning that the Enterprise Service Bus will make a call-back to this node. This signals the **exceptionTransition**.

Service2 has two outgoing transitions: the **ok** transition will be taken when things are happening normally, whilst the **exception** transition will be taken when the service has, as its name indicates, thrown an exception during processing:

```
<node name="Service2">
  <action class=
  "org.jboss.soa.esb.services.jbpm.actionhandlers.EsbActionHandler">
     <esbCategoryName>MockCategory</esbCategoryName>
     <esbServiceName>MockService</esbServiceName>
     <exceptionTransition>exception</exceptionTransition>
   </action>
   <transition name="ok" to="Service3"></transition>
   <transition name="exception" to="ExceptionHandling"/>
</node>
```

In the preceding definition of **Service2**, the action's exceptionTransition is set to **exception**. In this scenario, the process itself also ends up in the **ExceptionHandling** node.

Exception Decision: Observe the configuration of **Service3** and the **exceptionDecision** node that follows it. **Service3** processes to a normal conclusion and the transition out of its node occurs as one would expect.

However, at some point during the service execution, an **errorCode** was set, and the **exceptionDecision** node checks if a variable of the same name has been called here:

```
<node name="Service3">
  <action class=
  "org.jboss.soa.esb.services.jbpm.actionhandlers.EsbActionHandler">
     <esbCategoryName>MockCategory</esbCategoryName>
     <esbServiceName>MockService</esbServiceName>
     <esbToBpmVars>
        <mapping esb="SomeExceptionCode" bpm="errorCode"/>
     </esbToBpmVars>
  </action>
  <transition name="ok" to="exceptionDecision"></transition>
</node>

<decision name="exceptionDecision">
   <transition name="ok" to="end"></transition>
   <transition name="exceptionCondition" to="ExceptionHandling">
      <condition>#{ errorCode!=void }</condition>
   </transition>
</decision>
```

The **esbToBpmVars** mapping element extracts the **errorCode** called **SomeExceptionCode** from the message's body and sets in the *JBPM context*. (This is assuming that the **SomeExceptionCode** is set.)

In the next node, named **exceptionDecision** , the **ok** transition is taken if processing is normal, but if a variable called **errorCode** is found in the JBPM context, the **exceptionCondition** transition is taken instead.

To configure the system in this way, you need to use the JBPM's *decision node* feature. It allows you to nest multiple transitions within a condition:

```
<condition>#{ errorCode!=void }</condition>
```

NOTE

To learn more about conditional transitions, refer to the *JBPM Reference Guide*.

Report a bug

7.27. LAUNCHING THE JBPM CONSOLE

1. Once the server has stopped, access the **JBPM Console** from this address: http://localhost:8080/jbpm-console/app/processes.jsf.

2. You can now use the JBPM Console to deploy and monitor processes and tasks. The **bpm_orchestration4** quick start demonstrates this feature.

WARNING

The **JbpmDS** data source is defined in the **jbpm-ds.xml** file. By default, it uses a **Hypersonic** database. Always change this to a production-quality database in a live environment.

NOTE

Make sure that every **jbpm.esb** deployment shares the same database instance. (This is so that the various Enterprise Service Bus nodes have access to the same processes definitions.)

Report a bug

7.28. JBPM DEPLOYMENT

Table 7.10. JBPM Deployment

Property	Description
jbpm.esb/META-INF	This directory contains the **deployment.xml** and **jboss-esb.xml** files.

deployment.xml	Specifies the two resource files the ESB archive depends on: the **jbossesb.esb** and the **JbpmDS** data source files. The information in these files is used to determine the order of deployment:
	`<jbossesb-deployment>`
	`<depends>jboss.esb:deployment=jbossesb.esb</depends>`
	`<depends>jboss.jca:service=DataSourceBinding,name=JbpmDS</depends>`
	`</jbossesb-deployment>`

Property	Description

jboss-esb.xml	This file deploys an internal service called **JBpmCallbackService**:
	
	`<services>`
	`  <service category="JBossESB-Internal"`
	`name="JBpmCallbackService"`
	`  description="Service which makes Callbacks into jBPM">`
	`  <listeners>`
	`  <jms-listener name="JMS-DCQListener"`
	`busidref="jBPMCallbackBus"`
	`maxThreads="1" />`
	`  </listeners>`
	`  <actions mep="OneWay">`
	`  <action name="action"`
	`class="org.jboss.soa.esb.services.jbpm.actions.JBpmCallback"/>`
	`  </actions>`
	`  </service>`
	`</services>`
	
	This internal service listens to the **jBPMCallbackBus**, which, by default, is set to be either **JBossMQ** (via the **jbmq-queue-service.xml** file) or **JBossMessaging** (via the **jbm-queue-service.xml** file.) The latter is a messaging provider for the *Java Message Service Queue*. Ensure that only one of these files is deployed in the **jbpm.esb** archive.

Report a bug

CHAPTER 8. JBPM 5 INTEGRATION

8.1. INTEGRATION CONFIGURATION

The jbpm5.esb deployment that ships with the JBoss ESB includes the jBPM 5 runtime, a datasource, a hibernate configuration and a JPA persistence configuration. The datasource is configured to use H2 as a backend database by default and is used to persist session and process information.

In production you will want change from H2 to a production strength database. All jbpm5.esb deployments should share the same database instance so that the various JBoss ESB nodes have access to the same processes definitions and instances.

The jBPM GWT console is a web application. It is not included, but it can be optionally downloaded from the customer portal and installed through the jBPM installer. Use the jbpm-5.2.0.Final installer as this is the version that has been tested against the JBoss ESB. You may need to make minor changes to the configuration of the JBPM GWT console, including changing the "context-root" in WEB-INF/jboss-web.xml from 'jbpm-console' to 'jbpm5-console' so that it does not conflict with the jbpm 3 version console which ships by default in the ESB.

Check the jBPM documentation to change the security settings for this application. This requires you to change some settings in the conf/login-config.xml. The console can be used for deploying and monitoring jBPM processes, but it can also be used for human task management. For different users, a customized task list will be shown and they can administer those tasks.

The jbpm5.esb/META-INF directory contains the deployment.xml and the jboss-esb.xml. The deployment.xml specifies the resources this esb archive depends on.

Report a bug

8.2. JBPM 5 CONFIGURATION

By default the persistence.xml is set to use the JTA transaction manager, as defined in the section:

```
<?xml version="1.0" encoding="UTF-8"?><jbossesb-deployment>
 <depends>jboss.esb:deployment=jbossesb.esb</depends>
 <depends>jboss.jca:name=jboss/datasources/jbpm5DS,service=DataSourceBindi
ng</depends>
</jbossesb-deployment>
```

Other settings are left to the default jBPM settings and Hibernate is used to create the database schema.

Additionally, you can view the jBPM console's activities at any time by going to http://localhost:8080/business-central

Report a bug

8.3. JBOSSESB TO JBPM 5

JBossESB can make calls into jBPM 5 using the Bpm5Processor action. This action uses the jBPM 5 command API to make calls into jBPM. The following jBPM commands have been implemented:

Table 8.1. jBPM 5 commands

Command	Description
`startProcess`	Start a new ProcessInstance given a process definition that was already deployed to jBPM.
`signalEvent`	Signal to an already-started process that an event has occurred.
`abortProcessInstance`	Cancel a ProcessInstance. That is, when an event comes in which should result in the cancellation of the entire ProcessInstance. This action requires some jBPM context variables to be set on the message, in particular the ProcessInstance Id.

There are two required action attributes:

1. name

   Required attribute. You are free to use any value for the name attribute as long as it is unique in the action pipeline.

2. class

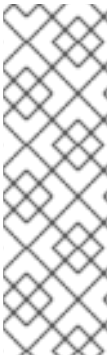
   Required attribute. This attributes needs to be set to "org.jboss.soa.esb.services.jbpm5.actions.Bpm5Processor"

WARNING

Using signalEvent is inherently risky because there is nothing to indicate what states the process instance or work item is in. Use the request-reply functionality of the ESBActionWorkItemHandler rather than signalEvent.

Report a bug

8.4. JBPM CONTEXT CONFIGURATION PROPERTIES



NOTE

The following properties can be set as so:

```
Message esbMessage = MessageFactory.getInstance().getMessage();

ContextImpl ctxi = (ContextImpl) esbMessage.getContext();
ctxi.setContext("jbpm5-session-id", 10);
ctxi.setContext("jbpm5-processinstance-id", 10L);
```

Table 8.2. jBPM Configuration Properties

Property	Description	Required?
process-action	Needs to be one of: startProcess, signalEvent, or abortProcessInstance.	Yes
process-definition-name	Required property. The value of this property should reference a process definition that is already deployed to jBPM and of which you want to create a new instance.	Yes
process-id	The value of this property should reference a process definition id in jBPM of which you want to create a new instance.	Yes
esbToBpmVars	Optional property. This property defines a list of variables that need to be extracted from the EsbMessage and set into jBPM context for the particular process instance. The list consists of mapping elements. Each mapping element can have the following attributes: • esb required attribute which can contain an MVEL expression to extract a value anywhere from the EsbMessage. • bpm optional attribute containing the name which be used on the jBPM side. If omitted the esb name is used. • value optional attribute which can hold a hard coded value.	No
handlerClass	WS Human Task handler class (default: org.jbpm.task.service.hornetq.CommandBasedHornetQWSHumanTaskHandler)	Yes
handlerHost	WS Human Task server hostname (default: 127.0.0.1)	Yes
handlerPort	WS Human Task server hostname (default: 5446)	Yes

NOTE

jBPM uses HornetQ by default. Another option is to use Mina which requires the following settings:

- org.jbpm.process.workitem.wsht.CommandBasedWSHumanTaskHandler
- handlerHost - WS Human Task server hostname (default: 127.0.0.1)
- handlerPort - WS Human Task server hostname (default: 9123)

Report a bug

8.5. BODY CONFIGURATION PROPERTIES

The following is a list of variables which can be set in context of the EsbMessage:

Table 8.3. Body Configuration Properties

Property	Description	Required?
jbpm5-processinstance-id	ESB context property that applies to the signalEvent and abortProcessInstance commands.	Yes
jbpm5-session-id	ESB context property that tells the action what session to load.	Yes

Report a bug

CHAPTER 9. SERVICE ORCHESTRATION AND THE ESB

9.1. SERVICE ORCHESTRATION

The term, service orchestration, refers to the arrangement of business processes. Traditionally, the Business Process Execution Language (BPEL) was used to execute SOAP-based web services. Red Hat recommends that you always use the JBPM to orchestrate your processes.

Report a bug

9.2. CREATING AN ORCHESTRATION DIAGRAM

1. Select **File** → **New** → **Other**.
2. Choose **`JBoss jBPM Process Definition`** from the **`Selection`** wizard.
3. Save the process definition. Use a separate directory for each process definition to avoid confusion.
4. Start to "drag-and-drop" items from the **jBPM Integrated Development Environment** 's menu palette into the **`Process Design`** view. You can switch between the design and source modes to check XML elements as you add them.
5. Add the XML fragments that are needed for the integration.
6. Before building the order process diagram, create and test the three services. These are ordinary ESB services and they are defined in the **`jboss-esb.xml`** file. An example configuration with service names and categories is shown below:

```
<services>
  <service category="BPM_orchestration4_Starter_Service"
  name="Starter_Service"
  description="BPM Orchestration Sample 4: Use this service to start
a
process instance">
   <!-- .... -->
  </service>
  <service category="BPM_Orchestration4" name="IntakeService"
  description="IntakeService: transforms, massages, calculates
priority">
   <!-- .... -->
  </service>
  <service category="BPM_Orchestration4" name="DiscountService"
    description="DiscountService">
  </service>
  <service category="BPM_Orchestration4" name="ShippingService"
    description="ShippingService">
   <!-- .... -->
  </service>
</services>
```

7. Refer to these services by using either the **EsbActionHandler** or **EsbNotifier** action handler. (Choose the **EsbActionHandler** in cases where the **JBoss Business Process Manager** expects a response, and the **EsbNotifier** when none is required.)

8. Now that the ESB services are known, drag the **Start** state node into the design view. A new process instance will begin at this node.

9. Drag in a node and name it **Intake Order**.

10. Connect the **Start** and the **Intake Order** nodes by selecting **Transition** from the menu and then clicking on each of them. An arrow connecting them will appear. It will be pointing towards the first **Intake Order**.

11. Add the Service and Category names to the Intake Node. Select the **Source** view. You will be able to see the source code for the **Intake Order** node. It should look like this:

```
<node name="Intake Order">
 <transition name="" to="Review Order"></transition>
</node>
```

12. Add the **EsbActionHandler** class reference, followed by the sub-element configuration settings for the service category and name, **BPM_Orchestration4** and the **IntakeService**. It should look like this:

```
<node name="Intake Order">
 <action name="esbAction" class=
  "org.jboss.soa.esb.services.jbpm.actionhandlers.EsbActionHandler">
   <esbCategoryName>BPM_Orchestration4</esbCategoryName>
   <esbServiceName>IntakeService</esbServiceName>
   <!-- async call of IntakeService -->
 </action>
<transition name="" to="Review Order"></transition>
</node>
```

13. Send some **JBoss Business Process Manager** context variables along with the service call using the following code. (In the example below, there is a variable named entireOrderAsXML which is to be set in the default position in the message body.)

```
<bpmToEsbVars>
 <mapping bpm="entireOrderAsXML" esb="BODY_CONTENT" />
</bpmToEsbVars>
```

This will cause the XML-based contents of the entireOrderAsXML variable to end up in the body of the message. Now the **IntakeService** can access the message and process it by letting it flow through each action in the pipeline. When the last action is reached, the replyTo property is checked and the message is sent to the **JBpmCallBack** service.

This makes a call back into the **JBoss Business Process Manager**, signaling the transition from the **Intake Order** node to the next one (in this case, **Review Order**).

14. Next, send some variables from the message to the node. Note that entire objects can be sent, as long both contexts can load the object's class. To retain the ability to "map back" to the **JBoss Business Process Manager**, add an esbToEsbVars element:

```
<node name="Intake Order">
<action name="esbAction" class=
"org.jboss.soa.esb.services.jbpm.actionhandlers.EsbActionHandler">
<esbCategoryName>BPM_Orchestration4</esbCategoryName>
<esbServiceName>IntakeService</esbServiceName>
<bpmToEsbVars>
<mapping bpm="entireOrderAsXML" esb="BODY_CONTENT" />
</bpmToEsbVars>
<esbToBpmVars>
<mapping esb="body.entireOrderAsXML" bpm="entireOrderAsXML"/>
<mapping esb="body.orderHeader" bpm="entireOrderAsObject" />
<mapping esb="body.customer" bpm="entireCustomerAsObject" />
<mapping esb="body.order_orderId" bpm="order_orderid" />
<mapping esb="body.order_totalAmount" bpm="order_totalamount" />
<mapping esb="body.order_orderPriority" bpm="order_priority" />
<mapping esb="body.customer_firstName" bpm="customer_firstName" />
<mapping esb="body.customer_lastName" bpm="customer_lastName" />
<mapping esb="body.customer_status" bpm="customer_status" />
</esbToBpmVars>
</action>
<transition name="" to="Review Order"></transition>
</node>
```

When this service returns, the following variables are then stored in the **JBoss Business Process Manager**'s context:

- **entireOrderAsXML**
- **entireOrderAsObject**
- **entireCustomerAsObject**

In addition, for demonstration purposes, there are also some flattened variables:

- **order_orderid**
- **order_totalAmount**
- **order_priority**
- **customer_firstName**
- **customer_lastName**
- **customer_status**

15. You must now review the order process manually. Add a **task node** with the task called **Order Review**. These jobs need to be performed by someone with the actor_id **user**.

    Make sure the XML fragment looks like this:

```
<task-node name="Review Order">
<task name="Order Review">
<assignment actor-id="user"></assignment>
<controller>
<variable name="customer_firstName"
```

```
access="read,write,required"></variable>
<variable name="customer_lastName" access="read,write,required">
<variable name="customer_status" access="read"></variable>
<variable name="order_totalamount" access="read"></variable>
<variable name="order_priority" access="read"></variable>
<variable name="order_orderid" access="read"></variable>
<variable name="order_discount" access="read"></variable>
<variable name="entireOrderAsXML" access="read"></variable>
</controller>
</task>
<transition name="" to="Calculate Discount"></transition>
</task-node>
```

16. Create an XHTML data form to display these variables in a form in the **jbpm-console**.

NOTE

See the *bpm_orchestration4* quick start's **Review_Order.xhtml** file for more information about this.

17. Link this data form to the task node by adding these settings to the **forms.xml** file:

```
<forms>
<form task="Order Review" form="Review_Order.xhtml"/>
<form task="Discount Review" form="Review_Order.xhtml"/>
</forms>
```

18. In this case, the same form is applied to two task nodes. There is a reference to the variables in the **Review Order** form as shown in the following sample code. (This, in turn, refers to the variables that are set in the **JBoss Business Process Manager**'s context.)

```
<jbpm:datacell>
<f:facet name="header">
<h:outputText value="customer_firstName"/>
</f:facet>
<h:inputText value="#{var['customer_firstName']}" />
</jbpm:datacell>
```

19. When the process reaches the **Review Node**, you can log into the jBPM Console and click on **Tasks** to see a list of items.

20. Click on a task to examine it in detail. A form will appear. You can then update some of the values.

21. Conclude by clicking **Save and Close**, at which point the process will move to the next node.

22. This one is the **Calculate Discount** node. It is, once again, an ESB service node, the configuration file for which looks like this:

```
<node name="Calculate Discount">
<action name="esbAction" class="
org.jboss.soa.esb.services.jbpm.actionhandlers.EsbActionHandler">
<esbCategoryName>BPM_Orchestration4</esbCategoryName>
```

```
<esbServiceName>DiscountService</esbServiceName>
<bpmToEsbVars>
<mapping bpm="entireCustomerAsObject" esb="customer" />
<mapping bpm="entireOrderAsObject" esb="orderHeader" />
<mapping bpm="entireOrderAsXML" esb="BODY_CONTENT" />
</bpmToEsbVars>
<esbToBpmVars>
<mapping esb="order"
bpm="entireOrderAsObject" />
<mapping esb="body.order_orderDiscount" bpm="order_discount" />
</esbToBpmVars>
</action>
<transition name="" to="Review Discount"></transition>
</node>
```

The service receives the **customer**, **orderHeader** and the **entireOrderAsXML** data. It then computes a discount. The response maps the **body.order_orderDiscount** value onto a **JBoss Business Process Manager** context variable called order_-discount. The process is signaled, which tells it to move to the **Review Discount** node.

23. Review the discount, which is set to a value of 8.5. Click on **Save and Close**. The process will move to the **Ship It** node, which is also an ESB service.

    To circumvent the order process before the **Ship It** service completes, use the **EsbNotifier** action handler by attaching it to the outgoing transition as shown below:

```
<node name="ShipIt">
<transition name="ProcessingComplete" to="end">
<action name="ShipItAction" class=
"org.jboss.soa.esb.services.jbpm.actionhandlers.EsbNotifier">
<esbCategoryName>BPM_Orchestration4</esbCategoryName>
<esbServiceName>ShippingService</esbServiceName>
<bpmToEsbVars>
<mapping bpm="entireCustomerAsObject" esb="customer" />
<mapping bpm="entireOrderAsObject" esb="orderHeader" />
<mapping bpm="entireOrderAsXML" esb="entireOrderAsXML" />
</bpmToEsbVars>
</action>
</transition>
</node>
```

    After notifying the **ShippingService**, the order process moves to the **end** state and terminates. (The **ShippingService** itself may still be finishing.)

    In the **bpm_orchestration4** quick start, the **JBoss Rules** engine is used to determine whether this order is to be shipped via the normal or the express method.

Report a bug

9.3. DEPLOYING A PROCESS DEFINITION

Once a **processdefinition.xml** file has been created, you can deploy it to the **JBoss Business Process Manager** using any of the following:

- the integrated development environment
- **ant**
- the jBPM Console

The following files will be deployed:

1. **Review_Order.xhtml**
2. **forms.xml**
3. **gpd.xml**
4. **processdefinition.xml**
5. **processimage.jpg**

The integrated development environment creates a **.PAR** archive and deploys it to the **jBPM**'s database.

WARNING

Red Hat advises you not to deploy Java code in **.PAR** archives as it may cause class-loading issues. Instead, use either **.JAR** or **.ESB** archives to deploy your classes.

Report a bug

9.4. INSTANTIATING A DEPLOYMENT

1. Create a new process instance once the process definition is deployed. (Note that you can use the **StartProcessInstanceCommand**. This command allows you to create a process instance with some pre-set initial values.)

```
<service category="BPM_orchestration4_Starter_Service"
name="Starter_Service"
description="BPM Orchestration Sample 4: Use this service to start a
process instance">
 <listeners>
 </listeners>
  <actions>
   <action name="setup_key" class=
    "org.jboss.soa.esb.actions.scripting.GroovyActionProcessor">
     <property name="script"
      value="/scripts/setup_key.groovy" />
  </action>
  <action name="start_a_new_order_process" class=
   "org.jboss.soa.esb.services.jbpm.actions.BpmProcessor">
```

```
    <property name="command"
     value="StartProcessInstanceCommand" />
    <property name="process-definition-name"
     value="bpm4_ESBOrderProcess" />
    <property name="key" value="body.businessKey" />
    <property name="esbToBpmVars">
     <mapping esb="BODY_CONTENT" bpm="entireOrderAsXML" />
   </property>
  </action>
 </actions>
</service>
```

2. The new process instance is now invoked and using a script. The jBPM key is set to the value of the OrderId by an incoming order XML file.

   This same XML is subsequently put into a **jBPM** context, through use of the **`esbToBpmVars`** mapping.

   In the **`bpm_orchestration4`** quick start, the XML came from the **`Seam DVD Store`** and the **`SampleOrder.xml`** looks like this:

```
<Order orderId="2" orderDate="Wed Nov 15 13:45:28 EST 2006"
statusCode="0"
 netAmount="59.97" totalAmount="64.92" tax="4.95">
  <Customer userName="user1" firstName="Rex" lastName="Myers"
state="SD"/>
   <OrderLines>
    <OrderLine position="1" quantity="1">
     <Product productId="364" title="Gandhi"
      price="29.98"/>
    </OrderLine>
    <OrderLine position="2" quantity="1">
      <Product productId="299" title="Lost Horizon" price="29.99"/>
    </OrderLine>
   </OrderLines>
</Order>
```

NOTE

Enterprise Service Bus and JBoss Business Process Manager deployments are what is known as "hot."

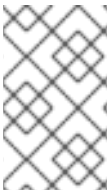
The jBPM has a special feature that results in process deployments being "versioned". Newly created process instances will use the latest version, while existing ones will run to their conclusion using the process deployment they were started on.

Report a bug

CHAPTER 10. SERVICE REGISTRY INTEGRATION WITH THE BPEL ENGINE

10.1. BPEL ENGINE

A BPEL engine executes BPEL business process instructions. The BPEL engine included as part of the JBoss Enterprise SOA Platform product is based on Apache ODE.



NOTE

It is recommended that you only open one BPEL console window in your browser. Failing to do so can result in seeing a blank window upon login or being unable to login from your second window. For details, see RIFTSAW-400.

Report a bug

10.2. 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

10.3. BPEL AND THE SERVICE REGISTRY

Because BPEL is integrated with the Service Registry, services can register themselves automatically as they are deployed.

This registration process utilizes the jUDDI client libraries. When a service is deployed, both it and its BindingTemplate (end-point reference) are registered and a partnerLinkChannel is created for each partnerLink. At the same time, the WSDL end-point is obtained from the UDDI.

Upon undeployment, the BindingTemplate is removed from the **UDDI Registry**.

Report a bug

10.4. ACTIVATE BPEL-SERVICE REGISTRY INTEGRATION

Procedure 10.1. Task

- Integration is turned on by default. To confirm this, open `vi SOA_ROOT/jboss-as/server/PROFILE/deploy/riftsaw.sar/bpel.properties.xml` and ensure that is set as follows: `bpel.uddi.registration=true`.

Report a bug

10.5. PARTNER LINK

A partner link is a link which establishes a relationship between a BPEL process and its client.

Report a bug

10.6. PARTNER LINK CHANNEL

A partner link channel is a communications channel that is used to interact with a client and the services integrated in a BPEL process.

Report a bug

10.7. ESB.JUDDI.CLIENT.XML

The **`SOA_ROOT/jboss-as/server/PROFILE/deploy/jbossesb.sar/esb.juddi.client.xml`** file is the client configuration file for the jUDDI Service Registry.

Report a bug

10.8. BPEL.PROPERTIES CONFIGURATION SETTINGS

Table 10.1. The UDDI-related properties in the bpel.properties file

attribute	type (default)	description
bpel.uddi.registration	boolean (true)	If set to 'false', the UDDI integration is turned off. The RiftSaw installation process sets this value to 'true' only if the jbossesb-registry.sar is detected containing a jUDDI v3 registry. In every other case it is automatically set to false.
bpel.webservice.secure	boolean (false)	The UDDI Registration process registers an WSDL AccessPoint in the BindingTemplate for the BPEL Service it is registering. The BPEL server exposes the service WSDL Endpoint on the WS stack (Currently Red Hat supports JBossWS and CXF). If your webservice stack is configured to use a secure protocol (such as https), you need to switch this setting to 'true'. (Note that this setting is used during the registration process only.)

attribute	type (default)	description
bpel.uddi.client.impl	String (org.jboss.soa.bpel.uddi.UDDIRegistrationImpl)	This is the name of the class that implements the org.jboss.soa.bpel.runtime.engine.ode.UDDIRegistration interface.
bpel.uddi.clerk.config	String (not used by default)	This defines the path to the **`bpel.uddi.client.xml`** configuration file. This can be left "commented out" if you want to use the **`riftsaw.sar/META-INF/riftsaw.uddi.xml`**. In this case, a **`bpel.uddi.clerk.manager`** must be defined.
bpel.uddi.clerk.manager	String (riftsaw-manager)	This defines the name of the ClerkManager that will be used if the riftsaw.uddi.xml is left commented out. This value should correspond to the name of the manager in the **`esb.juddi.client.xml`**. Note that if the bpel.uddi.clerk.config is defined, the bpel.uddi.clerk.manager setting is ignored.
bpel.uddi.clerk	String (BPEL_clerk)	This defines the name of the clerk that will be used. This value should correspond to the name of the clerk in the **`riftsaw.uddi.xml`**. (By default this is set to BPEL_clerk.)

attribute	type (default)	description
bpel.uddi.lookup	boolean (true)	If this is set to true, the creating process of the partner channel will do a lookup by serviceName in the UDDI, and a WSDL Endpoint is retrieved. This process makes it easier to move process deployment around within your server farm, without having to update the partnerlink WSDL files in your BPEL process deployments. If more than one end-point (BindingTemplate) is found, the default policy used by the ServiceLocator is 'PolicyLocalFirst'. Note that it is still a requirement to deploy the initial partnerlink WSDL file for each partnerLink.

NOTE

The names of both the ClerkManager and the Clerk itself are specified in the **`bpel.properties`** file.

Report a bug

10.9. CLERK

The clerk (**`org.apache.juddi.v3.client.config.UDDIClerk`**) is responsible for registering service end-points in the Service Registry.

Report a bug

10.10. SET THE PROPERTIES TO BE USED BY THE CLERK WHEN REGISTERING SERVICES

Procedure 10.2. Task

1. Open the **`esb.juddi.client.xml`** file in your text editor: **`vi SOA_ROOT/jboss-as/server/PROFILE/deploy/jbossesb.sar/esb.juddi.client.xml`**
2. Configure the settings. For example:

```
</nodes>
   <clerks registerOnStartup="false">
      <clerk name="SOAExample" node="default" publisher="root"
password="root"/>
```

```
    </clerks>
    </manager>
</uddi>
```

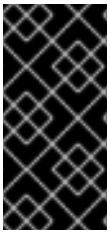
3. Save the file and exit.
4. Put another copy of the file in here (the files must always correspond: **SOA_ROOT/jboss-as/server/PROFILE/deploy/jbossesb-registry.sar/juddi_custom_install_data/**
5. Save the file and exit.

Report a bug

10.11. DEFAULT SETTINGS FOR THE SERVICE REGISTRY CLERK

Table 10.2. Default Settings

Property	Value
keyDomain	esb.jboss.org
businessKey	redhat-jboss
serviceDescription	BPEL Service deployed by Riftsaw
bindingDescription	BPEL Endpoint deployed by Riftsaw



IMPORTANT

The **SOA_ROOT/jboss-as/server/PROFILE/deploy/jbossesb-registry.sar/esb.juddi.xml** file contains a property called juddi.seed.always which is set to false. This means that it is always trying to load the root seed data when the server starts.

Report a bug

10.12. UDDI REGISTRATION

Upon deployment of a BPEL process, the process information is registered to the UDDI registry according to the BPEL4WS OASIS technote (http://www.oasis-open.org/committees/uddi-spec/doc/tn/uddi-spec-tc-tn-bpel-20040725.htm).

Report a bug

10.13. UDDI END-POINT LOOK-UP

If a BPEL service invokes another BPEL service (or a web service end-point in general), the BPEL Engine performs a lookup by serviceQName and portName (obtained from the WSDL). The result is stored in a client-side service cache, resulting in increased performance. To prevent the client-side cache from returning "stale" information, the cache is automatically invalidated by the UDDI registry using the Subscription API whenever changes occur in the registry.

Report a bug

PART IV. MESSAGE ROUTING

CHAPTER 11. USING RULES TO PERFORM CONTENT-BASED ROUTING

11.1. 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

11.2. INTRODUCING CONTENT-BASED ROUTING WITH ESB

Normally, data in the Enterprise Service Bus is packaged, transferred and stored in the form of a message. Messages are addressed to endpoint references (which point to either services or clients.) An endpoint reference's role is to identify the machine or process or object that will ultimately deal with the content of the message. However, what happens if the specified address is no longer valid? Situations that may lead to this scenario include those in which the service has failed or been removed.

It is also possible that the service no longer deals with messages of that particular type, in which case presumably some other service will still deal with the original function, but that still leaves the question of "How should the message be handled?" What if other services besides that which is the intended recipient are interested in the message's contents? What if no destination is specified?

This is where content-based routing comes in. The way that content-based routing works, a message is routed by being opened and then having a set of rules applied to its content. These rules are used to ascertain which parties are interested in it, allowing the ESB to determine the destination where it should be sent. This relieves the sending application of the need to know where the message should go.

Content-based routing systems are built around two components: routers (of which there may be only one) and services (of which there are normally more than one). Services are the components that ultimately "consume" the messages. The way in which each service indicates its interest in specific types of messages to the router is implementation-dependent, but some mapping must exist between the message type (or some other aspect of the message content) and the services in order for the router to direct them appropriately.

Routers, as their name suggests, "route" messages. They examine the content of messages as they receive them, apply rules to that content and then forward the messages as the rules dictate.

In addition to routers and services, some systems also include harvesters. The role of these tools is find interesting information, package it in the guise of a formatted message and send it to a router. Harvesters "mine" many sources of information, including mail transfer agent message stores, news servers, databases and other legacy systems.

Report a bug

11.3. DEFINING INLINE RULES FOR CONTENT-BASED ROUTING WITH XPATH

Procedure 11.1. Task

1. Open **jboss-esb.xml** and set the cbrAlias property to **XPath**.
2. Define the routing rules in the route-to configurations (found in the container destinations property) as shown:

```
<action class="org.jboss.soa.esb.actions.ContentBasedRouter"
name="ContentBasedRouter">
 <property name="cbrAlias" value="XPath"/>
 <property name="destinations">
  <route-to service-category="BlueTeam"  service-name="GoBlue"
expression="/Order[@statusCode='0']" />
  <route-to service-category="RedTeam"   service-name="GoRed"
expression="/Order[@statusCode='1']" />
  <route-to service-category="GreenTeam" service-name="GoGreen"
expression="/Order[@statusCode='2']" />
 </property>
</action>
```

Report a bug

11.4. DEFINING EXTERNAL RULES FOR CONTENT-BASED ROUTING WITH XPATH

Procedure 11.2. Task

1. Open the **jboss-esb.xml** file and set the cbrAlias property to **XPath**.
2. Define the routing expressions in a .properties file. Make sure the property keys correlate with the destination names and that the property values are the XPath expressions for routing to this destination.
3. Define the routing rules in the route-to configurations via the container destinations property. The destination-name attribute will refer to the XPath rule key as defined in the external .properties file as shown:

```
<action class="org.jboss.soa.esb.actions.ContentBasedRouter"
name="ContentBasedRouter">
   <property name="cbrAlias" value="XPath"/>
   <property name="ruleSet" value="/rules/xpath-rules.properties"/>
   <property name="ruleReload" value="true"/>
   <property name="destinations">
       <route-to destination-name="blue"  service-
category="BlueTeam"  service-name="GoBlue" />
       <route-to destination-name="red"   service-
category="RedTeam"   service-name="GoRed" />
```

```
        <route-to destination-name="green" service-
category="GreenTeam" service-name="GoGreen" />
    </property>
</action>
```

Report a bug

11.5. XPATH RULES FOR CONTENT-BASED ROUTING

The XPath rules are in a .properties file. They are represented by this syntax:

```
blue=/Order[@statusCode='0']
red=/Order[@statusCode='1']
green=/Order[@statusCode='2']
```

Report a bug

11.6. NAMESPACE

A namespace is a container that holds various identifiers. They define XML namespace prefix-to-URI (universal resource identifier) mappings which aid in providing requests to the server.

Report a bug

11.7. DEFINING XML NAMESPACE PREFIX-TO-URI MAPPINGS

- Define XML namespace prefix-to-URI mappings as shown below. (This applies to both external and in-line rule definitions.)

```
<action class="org.jboss.soa.esb.actions.ContentBasedRouter"
name="ContentBasedRouter">
    <property name="cbrAlias" value="XPath"/>
    <property name="namespaces">
        <route-to prefix="ord" uri="http://www.acne.com/order" />
    </property>
    <property name="destinations">
        <route-to service-category="BlueTeam"
        service-name="GoBlue"
expression="/ord:Order[@statusCode='0']" />
        <route-to service-category="RedTeam"
        service-name="GoRed"
expression="/ord:Order[@statusCode='1']" />
        <route-to service-category="GreenTeam"
        service-name="GoGreen"
expression="/ord:Order[@statusCode='2']" />
    </property>
</action>
```

Report a bug

11.8. DEFINING INLINE RULES FOR CONTENT-BASED ROUTING WITH REGEX

1. Open the file **jboss-esb.xml** and set the cbrAlias property to **Regex**.
2. Define the routing rules in the **route-to** configurations. (These are found in the container destinations property.)

```
<action class="org.jboss.soa.esb.actions.ContentBasedRouter"
name="ContentBasedRouter">
   <property name="cbrAlias" value="Regex"/>
   <property name="destinations">
       <route-to service-category="BlueTeam"  service-name="GoBlue"
expression="#*111#*" />
       <route-to service-category="RedTeam"   service-name="GoRed"
expression="#*222#*" />
       <route-to service-category="GreenTeam" service-
name="GoGreen" expression="#*333#*" />
   </property>
</action>
```

Report a bug

11.9. DEFINING EXTERNAL RULES FOR CONTENT-BASED ROUTING WITH REGEX

1. Open the file **jboss-esb.xml** and set the cbrAlias property to Regex.
2. Define the routing expressions in a .properties file. The property keys should be the destination names and the property values are the Regex expressions for routing to the destination.
3. Define the routing rules in the **route-to** configurations (found in the container destination property) with the destination-name attribute set to the Regex rule key as defined in the .properties file.

```
<action class="org.jboss.soa.esb.actions.ContentBasedRouter"
name="ContentBasedRouter">
   <property name="cbrAlias" value="XPath"/>
   <property name="ruleSet" value="/rules/regex-rules.properties"/>
   <property name="ruleReload" value="true"/>
   <property name="destinations">
       <route-to destination-name="blue"  service-
category="BlueTeam"  service-name="GoBlue" />
       <route-to destination-name="red"   service-
category="RedTeam"   service-name="GoRed" />
       <route-to destination-name="green" service-
category="GreenTeam" service-name="GoGreen" />
   </property>
</action>
```

The XPath rules are in a .properties file and are represented in this format:

```
blue=#*111#*
red=#*222#*
green=#*333#*
```

Report a bug

11.10. CONTENT BASED ROUTING USING THE JBOSS RULES ENGINE

JBoss Rules is the rule provider "engine" for the content-based router. The Enterprise Service Bus integrates with this engine through three different routing **`action classes`**, these being:

- a routing rule set, written in the **`JBoss Rules`** engine's DRL language (alternatively, you can use the DSL language if you prefer it);
- the message content. This is the data that goes into the JBoss Rules engine (it comes in either XML format or as objects embedded in the message);
- the destination (which is derived from the resultant information coming out of the engine).

NOTE

When a message is sent to the **`content-based router`**, a rule-set will evaluate its content and return a set of service destinations.

- **org.jboss.soa.esb.actions.ContentBasedRouter**: This action class implements the *content-based routing* pattern. It routes a message to one or more destination services, based on the message content and the rule set against which it is evaluating that content. The content-based router throws an exception when no destinations are matched for a given rule set or message combination. This action will terminate any further pipeline processing, so always position it last in your pipeline.
- **org.jboss.soa.esb.actions.ContentBasedWiretap**: This implements the *WireTap* pattern. The **`WireTap`** is an enterprise integration pattern that sends a copy of the message to a control channel. The **`WireTap`** is identical in functionality to the standard content-based router, however it does not terminate the pipeline. It is this latter characteristic which makes it suitable to be used as a wire-tap, hence its name. For more information, see http://www.eaipatterns.com/WireTap.html.
- **org.jboss.soa.esb.actions.MessageFilter**: This implements the *message filter* pattern. The message filter pattern is used in cases where messages can simply be dropped if certain content requirements are not met. In other words, it functions identically to the content-based router except that it does not throw an exception if the rule set does not match any destinations, it simply filters the message out. For more information, see http://www.eaipatterns.com/Filter.html.

Report a bug

11.11. XPATH DOMAIN-SPECIFIC LANGUAGE

NOTE

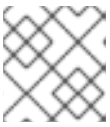
You may find it convenient to undertake an XPath-based evaluation of XML-based messages. Red Hat supports this by shipping a domain-specific language implementation. Use this implementation to add XPath expressions to the rule file.

1. First, define the expressions in the **`XPathLanguage.dsl`** file and use the following code to reference it in the rule set:

```
expander XPathLanguage.dsl
```

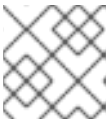
2. The XPath Language makes sure the message is in **`JBOSS_XML`** and that the following items have been defined:

    1. **`xpathMatch`**<*element*> : this yields **`true`** if an element by this name is matched.
    2. **`xpathEquals`**<*element*> , <*value*> : this yields **`true`** if the element is found and its value equals the value.
    3. **`xpathGreaterThan`**<*element*> , <*value*> : this yields **`true`** if the element is found and its value is greater than the value.
    4. **`xpathLessThan`**<*element*> , <*value*> : this yields **`true`** if the element is found and its value is lower then the value.



NOTE

The **`fun_cbr`** quick-start demonstrates this use of XPath.



NOTE

It is possible to define a completely different domain-specific language.

Report a bug

11.12. XPATH AND NAMESPACES

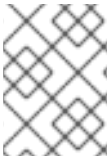
An XPath expression is a feature that searches through messages and extracts data from them.

To use namespaces in Xpath, specify which namespace prefixes are to be used in the XPath expression. These prefixes are specified in a comma-separated list in the following format: **`prefix=uri,prefix=uri`**.

`XPath Namespace-aware Statements:`

1. **`xpathMatch expr "<expression>" use namespaces "<namespaces>"`**
2. **`xpathEquals expr "<expression>", "<value>" use namespaces "<namespaces>"`**
3. **`xpathGreaterThan expr "<expression>", "<value>" use namespaces "<namespaces>"`**

4. `xpathLowerThan expr "<expression>", "<value>" use namespaces "<namespaces>"`



NOTE

Make sure these expressions have the **expr** at the beginning to stop them from clashing with non-XPath aware statements.

Report a bug

11.13. CONFIGURATION FOR CONTENT-BASED ROUTING

- XPath statements are connected through configuration settings stored in the **jboss-esb.xml** file. The **service configuration** below shows an example service configuration fragment. (In this example, the service is listening to a Java Message Service queue.)

```
<service>
         category="MessageRouting"
         name="YourServiceName"
         description="CBR Service">
         <listeners>
                  <jms-listener name="CBR-Listener"
                           busidref="QueueA" maxThreads="1">
            </jms-listener>
               </listeners>
  <actions>
       <action class="org.jboss.soa.esb.actions.ContentBasedRouter"
                name="YourActionName">
                     <property name="ruleSet"
value="JBossESBRules.drl"/>
                     <property name="ruleReload" value="true"/>
                     <property name="destinations">
                           <route-to destination-name="xml-
destination"
        service-category="category01"
              service-name="jbossesbtest1" />
                           <route-to destination-name="serialized-
destination"
                                              service-
category="category02"
        service-name="jbossesbtest2" />
                     </property>
                     <property name="object-paths">
    <object-path esb="body.test1" />
    <object-path esb="body.test2" />
        </property>
                   </action>
  </actions>
           </service>
```

Each message is passed to the **ContentBasedRouter** action class, which is loaded with a certain rule-set. It then sends the message to the JBoss Rules engine's working memory, runs the rules, obtains the list of destinations and sends copies of the message to the services.

In this case, it uses the **JBossESBRules.drl** rule-set which matches two destinations- **xml-destination** and **serialized-destination** . These names are mapped to those of real services in the **route-to** section.

Report a bug

11.14. CONTENT-BASED ROUTING ACTION TAG ATTRIBUTES

Table 11.1. Content-Based Routing Action Tag Attributes

Attribute	Description
Class	The action class, this being one of : **org.jboss.soa.esb.actions.ContentBasedRouter** , **org.jboss.soa.esb.actions.ContentBasedWiretap** or **org.jboss.soa.esb.actions.MessageFilter**
Name	A custom action name.

Report a bug

11.15. CONTENT-BASED ROUTING ACTION CONFIGURATION PROPERTIES

Table 11.2. Content-Based Routing Action Configuration Properties

Property	Description
ruleSet	This is the name of the file containing the **JBoss Rules engine's** **ruleSet** , which is the set of rules used to evaluate message content. (Only one **ruleSet** can be given for each content-based routing instance.)
ruleLanguage	This is an optional reference to a file containing the definition of a Domain Specific Language to be used for evaluating the rule set.
ruleReload	This is an optional property which can be set to **true** in order to enable "hot" redeployment of rule sets. Note that this feature will cause some overhead on the rules processing. Note also that the rules will reload if the **.esb** archive in which they reside is redeployed.
stateful	This is an optional property which tells the RuleService to use a stateful session where facts will be remembered between invocations. See the “Stateful Rules” section for more information about this topic.
destinations	This is a set of route-to properties, each of which contains the logical name of the destination, along with the Service category and name as referenced in the registry. The logical name is the name which should be used in the rule set.

Property	Description
object-paths	This is an optional property to pass **`message`** objects into **`working memory`**.
ruleAuditType	This is an optional property which allows the JBoss Rules engine to perform audit logging. The log can be read into the JBoss Developer Studio plug-in and inspected. Valid values are CONSOLE, FILE and THREADED_FILE. The default is that no audit logging will be performed.
ruleAuditFile	This is an optional property that allows you to to define the file-path for audit logging. Note that it only applies to FILE or THREADED_FILE ruleAuditType. The default is "event". Note that the JBoss Rules engine will append ".log" for you. The default location for this file is "." - the current working directory (which for JBoss is in its bin/ directory).
ruleAuditInterval	This is an optional property that allows you to define how often to flush audit events to the audit log. Note that this only applies to the THREADED_FILE ruleAuditType. The default is 1000 (milliseconds).

Report a bug

11.16. USING PRE-COMPILED RULE PACKAGES

The **`KnowledgeAgent`** is a component which is embedded in the JBoss Rules 5.0 API. No additional components are required to use the Knowledge Agent. If you are using the JBoss Enterprise BRMS Platform, the application only needs to include the drools-core dependencies in its classpath, i.e. the drools and mvel JARs only. There are no other rule-specific dependencies.

Use the **`KnowledgeAgent`** for pre-compiled rules packages. These packages can be on the local file system or in a remote location (accessed via a URL). Once you have built your rules in a package in the BRMS Platform (or from the ant task), you are ready to use the agent in your target application.

The following example constructs an agent that will build a new knowledge base from the files specified in the path String. It will poll those files every 60 seconds, which is the default, to see if they are updated. If new files are found it will construct a new KnowledgeBase. If the change set specifies a resource that is a directory it's contents will be scanned for changes too.

```
KnowledgeAgent kagent = KnowledgeAgentFactory.newKnowledgeAgent( "MyAgent"
);
kagent.applyChangeSet( ResourceFactory.newUrlResource( url ) );
KnowledgeBase kbase = kagent.getKnowledgeBase();
```

The KnowledgeAgent can accept a configuration that allows for some of the defaults to be changed. An example property is "drools.agent.scanDirectories", by default any specified directories are scanned for new additions, it is possible to disable this.

```
KnowledgeBase kbase = KnowledgeBaseFactory.newKnowledgeBase();

KnowledgeAgentConfiguration kaconf =
KnowledgeAgentFactory.newKnowledgeAgentConfiguration();
kaconf.setProperty( "drools.agent.scanDirectories", "false" ); // we don't
```

```
scan directories, only files

KnowledgeAgent kagent = KnowledgeAgentFactory.newKnowledgeAgent( "test
agent", kaconf );
// resource to the change-set xml for the resources to add
kagent.applyChangeSet( ResourceFactory.newUrlResource( url ) );
```

This is an example of a **change-set.xml**.

```
<change-set xmlns='http://drools.org/drools-5.0/change-set'";
    xmlns:xs='http://www.w3.org/2001/XMLSchema-instance'
    xs:schemaLocation='http://drools.org/drools-5.0/change-set drools-
change-set-5.0.xsd' >
    <add>
        <resource source='http://localhost:9000/TEST.pkg' type='PKG' />
    </add>
</change-set>
```

Resource scanning is enabled by default. It is a service and must be started, the same is for notification. This can be done via the ResourceFactory.

```
ResourceFactory.getResourceChangeNotifierService().start();
ResourceFactory.getResourceChangeScannerService().start();
```

The deployment screen of the **BRMS UI** provides URLs and downloads for packages. You need the Package URI's URL to include in the **change-set.xml** file so as to specify that you want this package. It specifies an exact version, in this case a snapshot. Each snapshot has its own URL. If you want the latest version then replace **NewSnapshot** with **LATEST**.

You can also download a package file **(PKG)** from the deployment screen's list of URLs. Put that file in a directory and use the **file** or **dir** feature of the **KnowledgeAgent**. This will automatically contact the **JBoss Enterprise BRMS Platform** server for updates which may not be wanted in some scenarios.

Report a bug

11.17. EXECUTING BUSINESS RULES

Executing rules that modify a message's contents according to business processes is very similar to executing rules for routing. An example quick-start called business_rule_service demonstrates this. (This quick-start makes use of the org.jboss.soa.esb.actions.BusinessRulesProcessor action class.)

This process makes use of a component called the Business Rule Processor. It is almost the same as a router, the only difference being that it returns the modified message to the action pipeline for further processing. You can even mix business and routing rules in a single rule-set if you wish to do so. (However, this will only work if one of those three routing action classes mentioned previously is used.)

Report a bug

11.18. USING YOUR OWN MESSAGING PROVIDER

To use your own messaging provider, modify the corresponding section in the **jboss-esb.xml** file to refer to it. Here is an example:

```
<providers>
  <jms-provider name="CallbackQueue-JMS-Provider"
    connection-factory="ConnectionFactory">
    <jms-bus busid="jBPMCallbackBus">
      <jms-message-filter dest-type="QUEUE"
        dest-name="queue/CallbackQueue" />
    </jms-bus>
  </jms-provider>
</providers>
```

Report a bug

PART V. MESSAGE TRANSFORMATION

CHAPTER 12. TRANSFORMATIONS WITH SMOOKS

12.1. SMOOKS

Smooks is a fragment-based data transformation and analysis tool. It is a general purpose processing tool capable of interpreting fragments of a message. It uses visitor logic to accomplish this. It allows you implement your transformation logic in XSLT or Java and provides a management framework through which you can centrally manage the transformation logic for your message-set.

Report a bug

12.2. USING SMOOKS

- Use the **`SmooksAction`** component to "plug" Smooks into an ESB action pipeline.

NOTE

You will find a number of quick-starts that demonstrate transformations in the **`samples/quick starts`** directory. (The name of each transformation of these quick starts is prefixed with the word **`transform_`**.)

Report a bug

12.3. OVERVIEW OF MESSAGE TRANSFORMATION WITH XSLT

This service works in the same way as Smooks and can be used as an alternative.

Report a bug

12.4. OVERVIEW OF MESSAGE TRANSFORMATION WITH ACTIONPROCESSOR DATA

If Smooks cannot handle a specific type of transformation, use this interface (**`org.jboss.soa.esb.actions.ActionPipelineProcessor`**) to implement a custom solution.

Report a bug

12.5. PROCESS TRANSFORMATION CONFIGURATION

This is an example of how to configure a transformation.

```
<actions mep="OneWay">
  <action class="org.jboss.soa.esb.actions.SystemPrintln" name="print-
before">
    <property name="message" value="[transform_XML2XML_simple] Message
before transformation">
```

```
    </property>
  <action class="org.jboss.soa.esb.smooks.SmooksAction" name="simple-
transform">
    <property name="smooksConfig" value="/smooks-res.xml">
    <property name="reportPath" value="/tmp/smooks_report.html">
    </property>
  <action class="org.jboss.soa.esb.actions.SystemPrintln" name="print-
after">
    <property name="message" value="[transform_XML2XML_simple] Message
after transformation">
    </property>
```

Line 1

mep stands for *message exchange pattern*. In this example, the requester invokes a service by sending it a message.

Lines 2-4

These configurations allow the message to be written to the server log before and after its transformation.

Line 5

This is where the SmooksAction is specified.

Line 6

This is where the XLST is specified.

Line 7

Generates a rep[ort of the transformation.

Report a bug

PART VI. MESSAGE PERSISTENCE

CHAPTER 13. MESSAGE PERSISTENCE

13.1. 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

13.2. MESSAGE STORE INTERFACE

This is the message store interface:

```
public interface MessageStore
{
    public MessageURIGenerator getMessageURIGenerator();
    public URI addMessage (Message message, String classification)
        throws MessageStoreException;
    public Message getMessage (URI uid)
        throws MessageStoreException;
    public void setUndelivered(URI uid)
        throws MessageStoreException;
    public void setDelivered(URI uid)
        throws MessageStoreException;
    public Map$lt;URI, Message> getUndeliveredMessages(String
classification)
        throws MessageStoreException;
    public Map$lt;URI, Message> getAllMessages(String classification)
        throws MessageStoreException;
    public Message getMessage (URI uid, String classification)
        throws MessageStoreException;
    public int removeMessage (URI uid, String classification)
        throws MessageStoreException;
}
```

Report a bug

13.3. FACTORS TO NOTE WHEN IMPLEMENTING A CUSTOM MESSAGE STORE

- Your implementation can use **addMessage** to derive a message classification scheme. If the classification is not defined, then it is up to the individual implementation of the **MessageStore** to determine for itself how it will store the message. Furthermore, the classification is only a guide: your implementation can ignore this field if need be.
- It is up to the implementation as to whether or not the **MessageStore** imposes any kind of *concurrency control* on individual messages so always use the **removeMessage** operation with care.
- Do not use the **setUndelivered** and **setDelivered** commands or other associated operations unless you are sure they are applicable. This is because the **MessageStore** interface supports both audit trail and re-delivery functionality.
- The **org.jboss.internal.soa.esb.persistence.format.db.DBMessageStoreImpl** class provides the default implementation of the message store. The methods in this implementation make the required database connections via a *pooled database manager*, called the **DBConnectionManager**.
- The **MessageActionGuide** and the **MessagePersister** actions override the message store implementation.
- The MessageStore interface does not currently support transactions. Any use of the message store within the scope of a global transaction will, therefore, be uncoordinated.

  The implication of this is that each **MessageStore** update or read will be undertaken separately and independently.

Report a bug

13.4. CONFIGURE A MESSAGE STORE

Procedure 13.1. 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 section and edit it to suit your configuration:

```
<properties name="dbstore">
  <!--  connection manager type -->
  <property name="org.jboss.soa.esb.persistence.db.conn.manager"
value=
"org.jboss.internal.soa.esb.persistence.manager.StandaloneConnection
Manager"/>
    <!-- this property is only used for the j2ee connection manager
-->
    <property
name="org.jboss.soa.esb.persistence.db.datasource.name"
      value="java:/JBossesbDS"/>
    <!-- standalone connection pooling settings -->
```

```
    <!--  mysql
    <property name="org.jboss.soa.esb.persistence.db.connection.url"
      value="jdbc:mysql://localhost/jbossesb"/>
    <property name="org.jboss.soa.esb.persistence.db.jdbc.driver"
      value="com.mysql.jdbc.Driver"/>
    <property name="org.jboss.soa.esb.persistence.db.user"
      value="kstam"/> -->
    <!--  postgres
    <property name="org.jboss.soa.esb.persistence.db.connection.url"
      value="jdbc:postgresql://localhost/jbossesb"/>
    <property name="org.jboss.soa.esb.persistence.db.jdbc.driver"
      value="org.postgresql.Driver"/>
    <property name="org.jboss.soa.esb.persistence.db.user"
      value="postgres"/>
    <property name="org.jboss.soa.esb.persistence.db.pwd"
      value="postgres"/> -->
    <!-- hsqldb -->
    <property name="org.jboss.soa.esb.persistence.db.connection.url"
      value="jdbc:hsqldb:hsql://localhost:9001/jbossesb"/>
    <property name="org.jboss.soa.esb.persistence.db.jdbc.driver"
      value="org.hsqldb.jdbcDriver"/>
    <property name="org.jboss.soa.esb.persistence.db.user"
value="sa"/>
    <property name="org.jboss.soa.esb.persistence.db.pwd" value=""/>
    <property
name="org.jboss.soa.esb.persistence.db.pool.initial.size"
      value="2"/>
    <property name="org.jboss.soa.esb.persistence.db.pool.min.size"
      value="2"/>
    <property name="org.jboss.soa.esb.persistence.db.pool.max.size"
      value="5"/>
    <!--table managed by pool to test for valid connections
        created by pool automatically -->
    <property
name="org.jboss.soa.esb.persistence.db.pool.test.table"
      value="pooltest"/>
    <property
name="org.jboss.soa.esb.persistence.db.pool.timeout.millis"
      value="5000"/>
</properties>
```

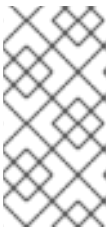
NOTE

You will find the scripts for the "required database" schema in the **SOA_ROOT/jboss-as/server/PROFILE/deploy/jbossesb.esb/message-store-sql/DB_TYPE/create_database.sql** file.

3. Sill in the global configuration file, configure the database connection manager:

```
<!--  connection manager type -->
<property name="org.jboss.soa.esb.persistence.db.conn.manager"
  value="org.jboss.internal.soa.esb.persistence.format.db.Standalone
ConnectionManager"/>
<!--  property name="org.jboss.soa.esb.persistence.db.conn.manager"
```

```
value="org.jboss.soa.esb.persistence.manager.J2eeConnectionManager"/
-->
<!-- this property is only used for the j2ee connection manager -->
<property name="org.jboss.soa.esb.persistence.db.datasource.name"
  value="java:/JBossesbDS"/>
```



NOTE

The Stand-Alone Manager uses C3PO to manage the connection pooling logic whilst the J2eeConnectionManager, by contrast, employs a data-source. Use the latter when deploying Enterprise Service Bus end-points inside a container such as a JBoss Application Server.

4. Save the file and exit.
5. Alternatively, you could plug in a custom connection manager by implementing the **`org.jboss.internal.soa.esb.persistence.manager.ConnectionManager`** interface and then updating the **`Properties`** file by providing it with the name of the new class.

Report a bug

13.5. CREATE_DATABASE.SQL

The **`SOA_ROOT/jboss-as/server/PROFILE/deploy/jbossesb.esb/message-store-sql/DB_TYPE/create_database.sql`** files contain scripts for database schemas.

Report a bug

13.6. CREATE_DATABASE.SQL SETTINGS

The following SQL code shows the structure of the **`create_database.sql`** files:

```
CREATE TABLE message
(
   uuid varchar(128) NOT NULL,
   type varchar(128) NOT NULL,
   message text(4000) NOT NULL,
   delivered varchar(10) NOT NULL,
   classification varchar(10),
   PRIMARY KEY (`uuid`)
);
```

- The UUID column is used to store a unique key for the message. This key takes the form of a standard uniform resource identifier.
- Message keys look like this:

```
urn:jboss:esb:message:UID: + UUID.randomUUID()_
```

NOTE

This syntax makes use of the UUID's random number generator.

- The "type" will be equal to that of the stored message. (The JBoss Enterprise SOA Platform ships with two different versions of "type", these being JBOSS_XML and JAVA_SERIALIZED, respectively.)
- The "message" column contains the contents of the actual message itself.
- The database message store implementation supplied works by invoking a connection to your pre-configured database. (Both a stand-alone connection manager, and another for using a JNDI data-source, are also supplied as part of the product.)
- The two pre-supplied connection managers for the database pool are: **`org.jboss.soa.esb.persistence.manager.StandaloneConnectionManager`** and **`org.jboss.soa.esb.persistence.manager.J2eeConnectionManager`**.

Report a bug

13.7. C3PO

C3P0 is an open source JDBC connection pool distributed along with Hibernate. It is found in the **`SOA_ROOT/jboss-as/lib`** directory.

Report a bug

13.8. J2EECONNECTIONMANAGER

The J2eeConnectionManager is a J2EE data-source-based connection pooling implementation for use when deploying to a J2EE container such as the JBoss AS application server.

Report a bug

13.9. JMSCONNECTIONPOOL

The JmsConnectionPool pools Java Message Service sessions. It is used by every JMS-based component, including the listeners, couriers and routers.

Report a bug

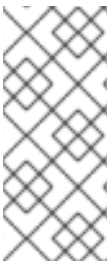
PART VII. CHANGE MANAGEMENT

CHAPTER 14. HOT DEPLOYMENT

14.1. 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

14.2. 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

14.3. 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

CHAPTER 15. VERSION MANAGEMENT

15.1. REDEPLOY A RULES FILE

Procedure 15.1. 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

15.2. REDEPLOY A TRANSFORMATION FILE

Procedure 15.2. 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

15.3. REDEPLOY A BUSINESS PROCESS DEFINITION

Prerequisites

- JBoss Developer Studio

Procedure 15.3. 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

15.4. RELOAD RULES WHILST RUNNING IN STANDALONE MODE

Procedure 15.4. Task

- Run **`ruleReload`**.

Report a bug

PART VIII. RULES SERVICES

CHAPTER 16. RULE SERVICE

16.1. RULE SERVICE

A rule service allows you to deploy business rules as services on the ESB. The service means that less client code is needed for integration. It also means rules can be processed from either an *action chain* or an *orchestrated business process*.

Report a bug

16.2. 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

16.3. STATEFUL RULES SESSIONS

Stateful rules sessions allow objects to be remembered across invocations in a session. To enable this, set the stateful set to **true**. You can tell stateful rule services when to continue with a current session and when to dispose of it.

Report a bug

16.4. STATEFUL RULES SESSION PROPERTIES

1. To continue the existing stateful session, set these two message properties:

```
message.getProperties().setProperty("dispose", false);
message.getProperties().setProperty("continue", true);
```

2. When you run the rules for the last time, set dispose to **true** to clear the JBoss Rules' engine's working memory:

```
message.getProperties().setProperty("dispose", true);
message.getProperties().setProperty("continue", true);
```

NOTE

To learn more about using stateful rules sessions, please refer to the **business_ruleservice_stateful** quick-start.

Report a bug

16.5. USING RULE SERVICES WITH JBOSS RULES

The JBoss Enterprise SOA Platform's rule service allows you to deploy business rules, written by your firm's business analyst in JBoss Rules, as services on the ESB. This has two major benefits:

- the amount of client code required to integrate the rules into your application environment is dramatically reduced.
- rules can be accessed from either an action chain or from within an orchestrated business process.

NOTE

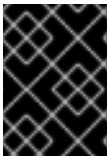
The JBoss Business Rules Management System is the supported option but you can also use a rule engine if you so desire.

The rule service functionality is provided by the **`BusinessRulesProcessor`** and **`DroolsRuleService`** action classes, the latter of which also implements the **`RuleService`** interface.

The **`BusinessRulesProcessor`** class allows rules to be loaded from the class-path. These rules are defined in **`.drl`** and **`.dsl`** files, and also in *decision tables* (which come in **`.xls`** format). These file-based rules exist primarily for the purpose of allowing you to test prototypes. There is no way to specify multiple rule files for a single **`BusinessRulesProcessor`** action. More complex rule services require the use of the **JBoss Rules `KnowledgeAgent`**.

The **`RuleService`** uses the **`KnowledgeAgent`** to access *rule packages* from either the **Business Rules Management System** or from the local file system. Rule packages can contain thousands of rules from different sources, including these:

1. the **JBoss Business Rules Management System**
2. imported **DRL** files
3. *domain-specific language* (DSL) files
4. *decision tables*



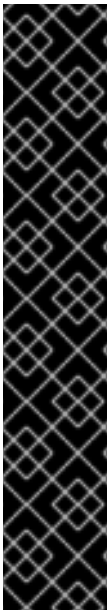
IMPORTANT

Red Hat recommends that you use the **`KnowledgeAgent`** approach on production systems.

The BusinessRulesProcessor action supports both of JBoss Rules' execution models, namely the *stateless* and *stateful* models.

Most rule services will adhere to the stateless model. In this model, a message will be sent to the rule service containing all the facts that are to be processed by the rule engine. The rules then execute and update the message and/or facts.

By contrast, a stateful execution takes place over a longer period time, with several messages being sent to the rule service. Each time a message is sent, the rules are executed again and the message and/or facts are updated. At the conclusion of the process, a final message tells the rule service to dispose of the stateful session's *working memory*, which is then purged from the rule engine.



IMPORTANT

RuleAgent is no longer used in the SOA Platform's ESB. It has been replaced by the KnowledgeAgent which is configured differently. The Polling configuration for DRL changes is no longer done via the **`ruleAgentProperties`** file's poll property). You now configure it globally via the org.jboss.soa.esb.services.rules.resource.scanner.interval property, found within **`esb.deployer/jbossesb-properties.xml`**. (The default value is 60.) This means that every sixty seconds, the system checks for resource changes across all KnowledgeAgents.

You will need to provide these properties to access URLs that are secured with basic authentication:

```
username=admin password=admin enableBasicAuthentication=true
```

NOTE

There can only be a single rule service in the message flow in the stateful model.

Report a bug

16.6. ACTION CHAIN

An action chain groups action components and executes them during a process.

Report a bug

16.7. ORCHESTRATED BUSINESS PROCESS

This kind of service service loosely couples applications for deployment so that they can be easily modified. It interacts with applications, services and components.

Report a bug

16.8. INTEGRATING JBOSS RULES AND THE SOA PLATFORM

JBoss Rules uses the following components to integrate with the SOA Platform.

`BusinessRulesProcessor`

An action class.

Rules

Any rules written in **JBoss Rules** , DRL, DSL, decision tables or the **Business Rule Editor** .

The Enterprise Service Bus message

This is inserted into the JBoss Rules Engine's working memory

The Enterprise Service Bus message's contents

This consists of the "fact" objects in the message, which are sent to the JBoss Rules Engine.

Report a bug

16.9. RULE SET REQUIREMENTS

- There are three requirements when creating rule sets on the **JBoss Enterprise SOA Platform.**

  The name and action class as shown:

  ```
  <action class="org.jboss.soa.esb.actions.BusinessRulesProcessor"
    name="OrderDiscountRuleService">
  ```

  One of the following is also required:

  - a **DRL** file:

    ```
    <property name="ruleSet" value="drl/OrderDiscount.drl" />
    ```

  - a **DSL** or **DSLR** (*Domain Specific Language*) file:

    ```
    <property name="ruleSet" value="dsl/approval.dslr" />
    <property name="ruleLanguage" value="dsl/acme.dsl" />
    ```

  - a **`decisionTable`** on the class-path:

    ```
    <property name="decisionTable" value="PolicyPricing.xls" />
    ```

  - a properties file on the class-path. This tells the **`rule agent`** how to find the rules package. You enable it by specifying either an URL or the path to a local file:

    ```
    <property name="ruleAgentProperties"
              value="brmsdeployedrules.properties" />
    ```

Report a bug

16.10. CREATING A RULE SET

1. Ensure the rule-set imports the ESB message. To test, use this code:

   ```
   import org.jboss.soa.esb.message.Message
   ```

2. Next, ensure the rule set defines the following global variable which creates the list of destinations:

   ```
   global java.util.List destinations;
   ```

The message is now sent to the JBoss Rules engine's working memory.

Now you can turn a regular rule set into one that can be used for content-based routing. Do this by evaluating the ESB message's matching rule and ensuring it outputs a list of *service destination names*.

Report a bug

16.11. RULE SET EXAMPLES

The rules are in a DRL file and execution following the stateless model:

```
      <action class="org.jboss.soa.esb.actions.BusinessRulesProcessor"
  name="OrderDiscountRuleService">
    <property name="ruleSet" value="drl/OrderDiscount.drl" />
    <property name="ruleReload" value="true" />
    <property name="object-paths">
        <object-path esb="body.Order" />
    </property>
</action>
```

The rules are in a DRL file and execution following the stateless model:

- The rules are in a DRL file and execution following the stateful model. In this scenario, the client can send multiple messages to the rule service. For example, the first message might contain a customer object, with the subsequent ones each containing orders for that customer. Every time a message is received, the rules are run. (The client can add a property to the final message that tells the rule service to dispose of the contents of the working memory.)

    1. A single, synchronized session instance is shared across all of the concurrent executions of a stateful session deployment. This greatly limits the number of use-cases for the stateful model. If you require multiple, client-oriented sessions service deployment, consider using a jBPM or BPEL solution instead.

    2. Stateful sessions are not *persistent* .

    3. Stateful Sessions are not *clustered* .

```
     <action
class="org.jboss.soa.esb.actions.BusinessRulesProcessor"
  name="OrderDiscountMultipleRuleServiceStateful">
    <property name="ruleSet"
              value="drl/OrderDiscountOnMultipleOrders.drl" />
    <property name="ruleReload" value="false" />
    <property name="stateful" value="true" >
    <property name="object-paths">
        <object-path esb="body.Customer" />
        <object-path esb="body.Order" />
    </property>
</action>
```

The rules are in a DRL file, the stateful model is used, audit logging is enabled and the JBoss Rules clockType , eventProcessingType and channels are configured to facilitate complex event processing .

Knowledge-base partitioning is not enabled since ruleMultithreadEvaluation is set to false (also note that ruleMaxThreads is set to 1).

NOTE

Complex event processing scenarios use the stateful model and are shared across all concurrent executions of a stateful session deployment.

```
     <action class="org.jboss.soa.esb.actions.BusinessRulesProcessor"
  name="OrderEventsRuleServiceStateful">
    <property name="ruleSet" value="drl/OrderEvents.drl" />
    <property name="ruleReload" value="false" />
    <property name="stateful" value="true" >
    <property name="ruleFireMethod" value="FIRE_UNTIL_HALT" />
    <property name="ruleAuditType" value="THREADED_FILE" />
    <property name="ruleAuditFile" value="myaudit" />
    <property name="ruleAuditInterval" value="1000" />
    <property name="ruleClockType" value="REALTIME" />
    <property name="ruleEventProcessingType" value="STREAM" />
    <property name="ruleMultithreadEvaluation" value="false" />
    <property name="ruleMaxThreads" value="1" />
    <property name="object-paths">
        <object-path esb="body.OrderStatus"
            entry-point="OrderStatusStream" />
        <object-path esb="body.OrderInfo"
            entry-point="OrderInfoStream" />
    </property>
    <property name="channels">
        <!-- chan1 and chan2 are equivalent
            (but timeout only applies if async is false) -->
        <send-to channel-name="chan1"
            service-category="cat1" service-name="svc1" />
        <send-to channel-name="chan2"
            service-category="cat1" service-name="svc1"
            channel-
class="org.jboss.soa.esb.services.rules.ServiceChannel"
            async="true" timeout="30000"
            set-payload-location="org.jboss.soa.esb.message.defaultEntry"
/>
        <!-- chan3 is a custom channel -->
        <send-to channel-name="chan3"
            channel-class="com.example.MyChannel" />
    </property>
</action>
```

The rules are in a Domain Specific Language format and the stateless execution model is employed:

```
      <action class="org.jboss.soa.esb.actions.BusinessRulesProcessor"
  name="PolicyApprovalRuleService">
    <property name="ruleSet" value="dsl/approval.dslr" />
    <property name="ruleLanguage" value="dsl/acme.dsl" />
    <property name="ruleReload" value="true" />
    <property name="object-paths">
        <object-path esb="body.Driver" />
        <object-path esb="body.Policy" />
```

```
    </property>
</action>
```

The rules are in a Decision Table and the stateless execution model is employed:

```
      <action class="org.jboss.soa.esb.actions.BusinessRulesProcessor"
  name="PolicyPricingRuleService">
    <property name="decisionTable"
              value="decisionTable/PolicyPricing.xls" />
    <property name="ruleReload" value="true" />
    <property name="object-paths">
        <object-path esb="body.Driver" />
        <object-path esb="body.Policy" />
    </property>
</action>
```

The rules are in a BRMS format, and the stateless execution model is employed:

```
      <action class="org.jboss.soa.esb.actions.BusinessRulesProcessor"
  name="RuleAgentPolicyService">
    <property name="ruleAgentProperties"
              value="ruleAgent/brmsdeployedrules.properties" />
    <property name="object-paths">
        <object-path esb="body.Driver" />
        <object-path esb="body.Policy" />
    </property>
</action>
```

Report a bug

16.12. BUSINESSRULESPROCESSOR ACTION CONFIGURATION ATTRIBUTES

Use these attributes to specify which action is to be undertaken and what name this action is to be given:

Table 16.1. Attributes

Attribute	Description
Class	Action class
Name	Custom action name

Report a bug

16.13. BUSINESSRULESPROCESSOR ACTION CONFIGURATION PROPERTIES

Table 16.2. BusinessRulesProcessor Action Configuration Properties

Property	Description
`ruleSet`	An optional reference to a file containing the **`ruleSet`** used to evaluate the content. Only one **`ruleSet`** can be given for each **`rule service`** instance.
`ruleLanguage`	An optional reference to a file containing the definition of a Domain Specific Language. This definition can be used for evaluating the rule set. Ensure that the file in the **`ruleSet`** is a **`dslr`**.
`ruleReload`	Set this optional property to **`true`** to enable the *hot redeployment* of **`rule sets`**. (Enabling this feature will increase the overhead on the rules processing.) Note that rules will reload if the **`.esb`** archive they reside in is redeployed.
`decisionTable`	An optional reference to a file containing the definition of a rule-specification spreadsheet.
stateful	Set this optional property to **`true`** to specify that the **`rule service`** will be receiving multiple messages over time. The rules will be re-executed each time.
object-paths	Optional property to pass message objects into JBoss Rules' working memory.
ruleFireMethod	Optional property to define the stateful rule execution method. Valid values are FIRE_ALL_RULES (default) and FIRE_UNTIL_HALT (launches its own thread. Useful for Complex Event Processing "CEP".)
ruleAuditType	Optional property to have Drools perform audit logging. The log can be read into the Drools Eclipse plugin and inspected. Valid values are CONSOLE, FILE and THREADED_FILE. The default is that no audit logging will be performed.
ruleAuditFile	Optional property to define the filepath for audit logging. Only applies to FILE or THREADED_FILE ruleAuditType. The default is "event". Note that JBoss Drools will append ".log" for you. The default location for this file is "." in the current working directory (which for JBoss is in its bin/ directory).
ruleAuditInterval	Optional property to define how often to flush audit events to the audit log. Only applies to the THREADED_FILE ruleAuditType. The default is 1000 (milliseconds).

Property	Description
ruleClockType	Optional property to define the clock used by JBoss Drools. Valid values are REALTIME and PSEUDO. The default is REALTIME.
ruleEventProcessingType	Optional property to define the event processing option used by Drools. The valid values are CLOUD or STREAM (useful for CEP). The default is CLOUD.
ruleMultithreadEvaluation	Optional property to define whether or not to enable KnowledgeBase partitioning. The default is null, which delegates to Drools' default (which is false).
ruleMaxThreads	Optional property to define the number of threads to use for KnowledgeBase partitioning. This is only respected if ruleMultithreadEvaluation is true. The default is null, which delegates to JBoss Rules' default.
channels	Optional property to define any Channels a rule can send Objects to. The Channel concept used to be known as an ExitPoint.

Report a bug

16.14. OBJECT PATHS

An object path contains information about the location of an object. It is used by applications to find these objects,

Report a bug

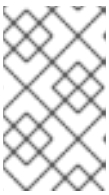
16.15. MVFLEX EXPRESSION LANGUAGE (MVEL)

MVEL is an MVFLEX Expression Language. This is a programming language which is used in Java environments.

Report a bug

16.16. USING OBJECT PATHS

1. Set the object-paths property to extract those objects with an **`ESB Message Object Path`**.



NOTE

You can use the object-paths property to drill down further into object trees. Do so by setting this property to extract those objects with an **ESB Message Object Path**.

2. The JBoss Rules engine uses the *MVFLEX Expression Language* (MVEL) to extract the object. Your path must abide by this syntax:

```
location.objectname.[beanname].[beanname]...
```

IMPORTANT

Remember to add the **`java import`** statements to any objects that you import into your rule set.

NOTE

The *Object Mapper* cannot "flatten out" entire collections. If you need to do that, run a "transformation" process on the message first, as this "unrolls" the collection.

Report a bug

16.17. OBJECT PATH PROPERTIES

Table 16.3. Object Path Properties

Property	Description
location	This must be one of the following: {body, header, properties, attachment}
objectname	The name of the object. (Attachments can be named or numbered, so for attachments this can be a number too.)
beannames	This is optional. Use this tag to traverse a bean graph.

Report a bug

16.18. MVEL EXPRESSIONS

Table 16.4. MVEL Expressions

Expression	Result
`properties.Order`	Use this to obtain the property object named **`Order`** .
`attachment.1`	Obtains the first attachment object.

Expression	Result
`attachment.AttachmentOne`	Obtains the attachment named `AttachmentOne`
`attachment.1.Order`	Obtains `getOrder()` *return object* on the attached object.
`body.Order1.lineitem`	Obtains the object named `Order1` from the body of the message. It will then call `getLineitem()` on this object. More elements can be added to the query in order to traverse the bean graph.

Report a bug

16.19. SENDING OBJECTS TO JBOSS RULES USING CHANNELS

1. To send an object to a JBoss Rules engine channel, use this DRL syntax. (It goes on the right-hand side of the *rule definition*.)

```
channels["mychannel"].send(myobject);
```

2. For the above code to work, register `mychannel` as your channel. To do this you add a channels property section to the `jboss-esb.xml` configuration file.

   Define as many channels as you want. For each particular channel name, the channels will be executed in the same order as they appear in the configuration file.

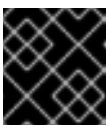
3. The following channels are supported:

   - The `ServiceChannel` (default). Specify the channel-name, service-category and service-name as attributes of the send-to element.

     There are also optional attributes, including async, timeout and set-payload-location (where the object sent to that channel will be placed in a new ESB message).

     This code shows you how to set an attribute:

```
<property name="channels">
    <send-to channel-name="mychannel"
        service-category="cat1" service-name="svc1" />
</property>
```

     IMPORTANT

     Make sure you have `invmScope="GLOBAL"` defined on the target service.

   - Specify the custom channels by using your own `org.drools.runtime.Channel` implementation class. The send-to attribute for this is channel-class. Your implementation requires a *public no-arg constructor*.

If you want your implementation to be configurable, implement the **`org.jboss.soa.esb.Configurable`** interface so your **`setConfiguration(ConfigTree()`** method will be called. This allows it to pass the attributes and sub- property elements to your custom channel.

This code sample shows how to configure a custom channel:

```
<property name="channels">
    <send-to channel-name="mychannel"
        channel-class="com.example.MyChannel" />
</property>
```

Report a bug

16.20. PACKAGING AND DEPLOYING RULES

1. Package your rule code into functional units by putting them in **`.esb`** packages. (The aim is to package routing rules alongside the **`rule services`** that use the **`rule sets`**.)

2. Once you have created the **`jbrules.esb`** archive deploy and reference it via the **`deployment.xml`** file. Use this code to do so:

```
<jbossesb-deployment>
 <depends>jboss.esb:deployment=jbrules.esb</depends>
</jbossesb-deployment>
```

Report a bug

PART IX. PROTOCOL TRANSLATION

CHAPTER 17. ADAPTERS

17.1. 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

17.2. PROVIDER ADAPTER

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

Report a bug

17.3. 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

17.4. 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

17.5. JCA ADAPTER

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

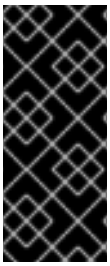
Report a bug

PART X. SECURITY

CHAPTER 18. SECURITY

18.1. 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

18.2. ENABLE SERIALIZATION ON THE SERVER

Procedure 18.1. 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

18.3. ENABLE SERIALIZATION ON THE CLIENT

Prerequisites

- Server serialization must already be enabled.

Procedure 18.2. 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

18.4. 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

18.5. 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

18.6. PER-SERVICE SECURITY PROPERTIES

Table 18.1. 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

18.7. OVERRIDE GLOBAL SECURITY SETTINGS

Procedure 18.3. 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

18.8. SECURITY PROPERTY OVERRIDES

Table 18.2. 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

18.9. 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

18.10. 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

18.11. 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

18.12. ADD AN AUTHENTICATION CLASS TO A MESSAGE OBJECT

Procedure 18.4. 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

18.13. 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

18.14. SET A TIME LIMIT FOR THE SECURITY CONTEXT GLOBALLY

Procedure 18.5. 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

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

Procedure 18.6. 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

18.16. SECURITY SERVICE

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

Report a bug

18.17. 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

18.18. SECURITYCONTEXTPROPAGATOR

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

Report a bug

18.19. SECURITYCONTEXTPROPAGATOR IMPLEMENTATIONS

Table 18.3. 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

18.20. ADD A CUSTOM LOG-IN MODULE

Procedure 18.7. 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

18.21. 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

18.22. 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 18.4. 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

18.23. 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 18.5. 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

18.24. 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

18.25. 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

18.26. ENABLE ROLE MAPPING

Procedure 18.8. 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

18.27. SECURITY_CERT QUICKSTART

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

Report a bug

18.28. CUSTOMIZE THE SECURITY SERVICE INTERFACE

Procedure 18.9. 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

18.29. 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

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

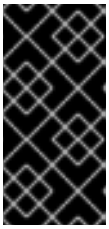
Client applications can, by default, utilize Remote Method Invocation to download Enterprise Java Bean classes through `port 8083`. However, you can also configure the system's Remote Method Invocation settings to allow client applications to download any deployed resources you desire.

Procedure 18.10. 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 19. SECURING THE SERVICE REGISTRY

19.1. 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

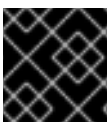
19.2. AUTHTOKEN

An authToken is a security container holding password credentials.

Report a bug

19.3. 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

19.4. OBTAIN AN AUTHTOKEN

Procedure 19.1. 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

19.5. 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

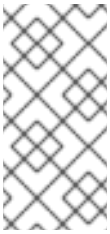
The **`CryptedXMLDocAuthentication`** implementation is similar to the **`XMLDocAuthentication`** implementation, but the passwords are encrypted:

```
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

19.6. CONFIGURE XMLDOCAUTHENTICATION

Procedure 19.2. 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

19.7. LIGHTWEIGHT DIRECTORY ACCESS PROTOCOL (LDAP)

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

Report a bug

19.8. CONFIGURE LDAP AUTHENTICATION

Procedure 19.3. 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

19.9. CONFIGURE JBOSS AUTHENTICATION

Procedure 19.4. 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

APPENDIX A. REVISION HISTORY

Revision 5.3.1-78.400 **2013-10-31** **Rüdiger Landmann**

Rebuild with publican 4.0.0

Revision 5.3.1-78 **Mon Feb 18 2013** **CS Builder Robot**

Built from Content Specification: 6713, Revision: 374431