JBoss Enterprise SOA Platform 5

JBoss SOA BPEL Guide

This guide is for developers

Edition 5.3.1

Last Updated: 2017-10-27

JBoss Enterprise SOA Platform 5 JBoss SOA BPEL Guide

This guide is for developers
Edition 5.3.1

David Le Sage
Red Hat Engineering Content Services

Suzanne Dorfield
Red Hat Engineering Content Services

Legal Notice

Abstract

This guide teaches developers how to use the Business Process Execution Language (BPEL) Engine.

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?

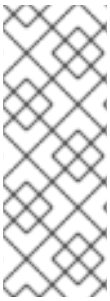
Introduction

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

CHAPTER 2. INTRODUCTION

2.1. INTENDED AUDIENCE

This book has been written for developers wanting to work with the JBoss Enterprise SOA Platform's BPEL Engine.

Report a bug

2.2. AIM OF THIS BOOK

Read this book in order to learn how to work with the JBoss Enterprise SOA Platform's Business Process Execution Language (BPEL) Engine. It is assumed that your system had already been installed and configured correctly as per the instructions in the *Installation and Configuration Guide*.

Report a bug

2.3. BACK UP YOUR DATA

WARNING

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

Report a bug

2.4. VARIABLE NAME: SOA_ROOT DIRECTORY

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

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

Report a bug

2.5. VARIABLE NAME: PROFILE

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

Report a bug

2.6. SERVER PROFILES

Table 2.1. Server Profiles

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

Report a bug

2.7. JAVA VIRTUAL MACHINE

A Java Virtual Machine (JVM) is a piece of software that is capable of running Java bytecode. The JVM creates a standard environment in which the intermediate bytecode is run. By creating the standard

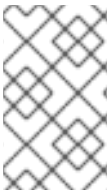
environment irrespective of the underlying hardware and operating system combination, it allows programmers to write their Java code once and have confidence that it can be run on any system. Red Hat recommends customers use OpenJDK as it is an open source, supported Java Virtual Machine that runs well on Red Hat Enterprise Linux systems. Windows users should install Oracle JDK 1.6.

Report a bug

CHAPTER 3. THE BPEL ENGINE

3.1. 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

3.2. 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

3.3. BUSINESS RULE ORCHESTRATION

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

Report a bug

3.4. ENTERPRISE SERVICE BUS

An enterprise service bus is a concrete implementation of an abstract SOA design philosophy. An enterprise service bus (ESB) has two roles: it provides message routing functionality and allows you to register services. The enterprise service bus that lies at the center of the JBoss Enterprise SOA Platform is called JBoss ESB.

An enterprise service bus deals with infrastructure logic, not business logic (which is left to higher levels). Data is passed through the enterprise service bus on its way between two or more systems. Message queuing may or may not be involved. The ESB can also pass the data to a transformation engine before passing it to its destination.

Report a bug

3.5. APACHE ODE

Apache ODE ("Orchestration Director Engine") is a software component that is designed to execute

BPEL business processes. It sends and receives messages to and from web services, manipulates data and performs error handling in the method prescribed in your process definition. To learn more about Apache ODE, visit the project website at http://ode.apache.org/.

Report a bug

3.6. PROCESS DEFINITION

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

Report a bug

3.7. PROCESS INSTANCE

A process instance is one execution of a process definition.

Report a bug

CHAPTER 4. AN INTRODUCTORY TUTORIAL

4.1. RUNNING BPEL PROCESSES ON THE JBOSS ENTERPRISE SOA PLATFORM

Introduction

The "BPEL ESB Hello World" quickstart demonstrates how an ESB service can directly invoke a BPEL process, provided that:

1. The BPEL engine and the ESB are located in the same JVM
2. The invoked BPEL process is deployed to the BPEL engine instance running on the same ESB.

Report a bug

4.2. DEPLOY THE "BPEL ESB HELLO WORLD" QUICKSTART

Procedure 4.1. Deploy the "BPEL ESB Hello World" Quickstart

1. Check that the JBoss Enterprise SOA Platform server is running.
2. Navigate to the directory containing the quickstarts: **`cd SOA_ROOT/samples/quickstarts/bpel_helloworld`** (or **`chdir SOA_ROOT\samples\quickstarts\bpel_helloworld`** in Microsoft Windows).
3. Deploy the quickstart by running **`ant deploy`**.
4. Execute the quickstart by running **`ant runtest`**.

Result

This command will send the text "Hello World via ESB to BPEL" to the ESB service, which will invoke the BPEL process. This in turn will append "Hello World" to the text and echo it back until it is received by the client application.

Report a bug

4.3. 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

4.4. ANT DEPLOY

`ant deploy` compiles a quickstart's source code in the **`build`** directory, then generates an .ESB file

(such as **Quickstart_helloworld.esb**) in the server profile's **deploy** directory. (Note that it generates .JAR files for BPEL quickstarts.) The server detects the presence of the new .esb archive and deploys it. In the .ESB archive is a deployment.xml file that **ant deploy** uses to configure a queue.

ant deploy also uses an XSL template to transform generic JMS queue names into the specific JMS queues needed by the target server's messaging provider. Ant selects the correct messaging provider by examining the server for a messaging provider deployment. Only JBoss Messaging, JBoss MQ, and HornetQ are detected by the build script. Other messaging providers are not supported by the quick start. Ant then puts the **deployment.xml** file into the **build/META-INF** directory before including it in the same .ESB archive as the rest of the quickstart.

Report a bug

4.5. ANT RUNTEST

ant runtest sends an ESB-unaware "Hello World" message (which is a plain String object) to the JMS Queue (**queue/quickstart_helloworld_Request_gw**). It instructs Java to run the sender class (in the case of the "Hello World" quick start, this is called **org.jboss.soa.esb.samples.quickstart.helloworld.test.sendJMSMessage**). By doing so, it sends a message directly to the deployed process.

Report a bug

CHAPTER 5. MANAGING PROCESSES

5.1. DEPLOYING PROCESSES

There are two ways in which you can deploy BPEL processes:

1. Via JBoss Developer Studio
2. Directly (the sample quick starts use this approach).

Report a bug

5.2. DEPLOY A PROCESS USING JBOSS DEVELOPER STUDIO

Prerequisites

- JBoss Developer Studio.

Procedure 5.1. Task

1. Launch JBoss Developer Studio.
2. Create or import the JBDS BPEL project.

   (In this example, we are going to import an existing project from the **SOA_ROOT/jboss-as/samples/quickstart/bpel_esb_helloworld** directory).

   Select the **Import** menu item (found in JBDS's left-side navigation panel-> **General->Existing Projects into Workspace->Next**.
3. Click the **Browse** button.
4. Navigate to the **bpel_esb_helloworld** quickstart directory.

   NOTE

   Once the project has been imported, you can inspect its contents, including the BPEL process and WSDL description.
5. Create a server configuration for the **JBoss Enterprise SOA Platform**. (The **Server** tab should be visible in the lower region of the JBDS window. If it is not present, click **Window->Show Views->Servers**.)
6. Right-click in the **Servers** view and select **New->Server**.
7. Select the appropriate version of the **JBoss Enterprise SOA Platform.**
8. Click **Finish**.
9. Start the new server by right-clicking on the server in the **Servers** tab and selecting **Start**.

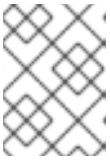
   The output from the server will be displayed in the **Console** tab.

10. Once the server has been started, right-click on the server entry again, and select **Add and Remove->bpel_esb_helloworld**.

11. Click **Add->Finish**.

    The project deploys to the server. (It will show up as an entry below the server in the **Servers** tab.)

12. Test the deployed BPEL process by using the **Ant** script provided with the quick start sample. Right-click on the project root folder's **build.xml** file and select **Run As->Ant Build**.

    NOTE

    It is important to select the menu item with the "**...**", as this launches a dialogue box that enables you to select which **Ant** target you wish to utilise.

13. De-select the **deploy** target.

14. Select the **runtest** target.

15. Click the **Run** button.

    A test 'hello' message is now sent to the server, and the response is displayed in the **Console** tab.

16. To update the BPEL process, select **assignHelloMesg->Properties->Details->Expression to Variable**.

    You can update the **concat** function's second parameter in order to perform tasks such as adding **UPDATED** to the text.

17. Save the update.

18. Go to the **Server** view.

19. Click **bpel_esb_helloworld>Full Publish**.

    The project redeploys.

20. Re-run the **runtest** target within the **build.xml** file in order to send a new request and view the new response.

    The response should now reflect the changes you made in the BPEL process.

21. Go to the **Server** view and expand the deployed project node.

    You will see that both of the deployed versions are shown. (The older version is retained as there may still be BPEL process instances using that version of the process.)

22. Use the **Server** view's project menu and click **Add and Remove** to undeploy the project.

WARNING

If you right-click on each of the child nodes, you will see it is also possible to undeploy the specific versions. However, if you do manually undeploy a version, any remaining active process instances for that version will be terminated.

23. Go to the **Server** menu item and stop the server.

Report a bug

5.3. TEST A PROCESS USING JBOSS DEVELOPER STUDIO

Prerequisites

- JBoss Developer Studio

Procedure 5.2. Task

1. Launch JBoss Developer Studio.
2. Right-click on the process and select **WSDL**.
3. Click on **Web Services**.
4. Click on **Test with Web Service Explorer** .
5. Launch JBoss Developer Studio.

Report a bug

5.4. DEPLOY A PROCESS USING THE DIRECT APPROACH

Procedure 5.3. Task

1. Open the process' Apache Ant build script in your text editor.
2. Modify it appropriately. Here is an example:

```
<!-- Import the base Ant build script... -->
 <property file="../../../install/deployment.properties" />

 <property name="version" value="1" />

 <property name="server.dir"
value="${org.jboss.as.home}/server/${org.jboss.as.config}"/>
 <property name="conf.dir" value="${server.dir}/conf"/>
```

```
 <property name="deploy.dir" value="${server.dir}/deploy"/>
 <property name="server.lib.dir" value="${server.dir}/lib"/>

 <property name="sample.jar.name" value="${ant.project.name}-
${version}.jar" />

 <target name="deploy">
  <echo>Deploy ${ant.project.name}</echo>
  <jar basedir="bpel" destfile="${deploy.dir}/${sample.jar.name}" />
 </target>

 <target name="undeploy">
  <echo>Undeploy ${ant.project.name}</echo>
  <delete file="${deploy.dir}/${sample.jar.name}" />
</target>
```

3. Save the file and exit.

Report a bug

5.5. SOME POINTS ABOUT DIRECT DEPLOYMENT OF PROCESSES

- It is necessary to provide the location of the JBoss Enterprise SOA Platform Server on which the BPEL process is to be deployed. (In this example, this is achieved by referring to the **deployment.properties** file found in the installation directory.
- You can take a *versioned approach* where multiple versions of the same BPEL process are deployed at once. The name of the **JAR** archive file containing the BPEL process and any associated artifacts will have a version number suffix. You will need to increment this suffix number manually every time you deploy a distinct version of the BPEL process.

WARNING

Currently, the version must be specified as a single integer number. Non-numeric values, such as those that appear in a major.minor.incremental (**Maven**-style) format, trigger an exception.

- You need to define the deployment target to be used to create the BPEL process archive. Do so by using the contents of the **bpel** sub-directory and store them within the JBoss Enterprise SOA Platform's **SOA_ROOT/jboss-as/server/PROFILE/deploy** directory.
- Finally, you need to define the undeployment target. This is simply used to remove the BPEL process archive from the **JBoss Enterprise SOA Platform**'s **SOA_ROOT/jboss-as/server/PROFILE/deploy** directory.

Report a bug

5.6. REMOVE A PROCESS

WARNING

If you remove ("undeploy") an active process definition, every one of its process instances will be terminated. If you do not want this to happen, then you must retire the process definition first.

Procedure 5.4. Task

1. Check to see if the processes have any active process instances.
2. If there are any, use the BPEL Web Console to retire them first.
3. Remove the processes only when there are no longer any active process instances.

Report a bug

5.7. CONFIGURE AND DEPLOY AN END-POINT REFERENCE

Procedure 5.5. Task

1. Create a sample end-point file: **`vi SOA_ROOT/jboss-as/samples/quickstart/hello_world/bpelContent/test.endpoint`**.
2. Add the following content to it:

```
# 3 minutes
mex.timeout=180000
```

3. Save the file and exit.

   Once deployed, the **`helloworld`** example will wait up to three minutes to respond.

4. To test this out, edit the **`hello_world.bpel`** file and insert a **`wait`** activity before the response:

```
<wait>
 <for>'PT150S'</for>
</wait>
```

   This will make it wait two minutes and thirty seconds before responding, which is thirty seconds longer than the default timeout, but still within the limits of the new timeout period specified within the **`test.endpoint`** file.

5. Save the file and exit.
6. If you wish to force the timeout to occur, increase the wait duration to three minutes and thirty

seconds and re-submit the test message using the `ant runtest` command.

NOTE

The BPEL Web Console does not yet support the global configuration file, so it can only obtain the configuration data from the deployed BPEL bundle. (Specifically, it looks in the root location within the BPEL deployment unit alongside the BPEL deployment descriptor.)

Report a bug

5.8. SUPPORTED END-POINT REFERENCE CONFIGURATION PROPERTIES

The BPEL Web Console supports the following properties:

- mex.timeout

Report a bug

CHAPTER 6. CONFIGURING WEB SERVICES

6.1. WEB SERVICE

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

Report a bug

6.2. WEB SERVICE END-POINT

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

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

Report a bug

6.3. WEB SERVICES DESCRIPTION LANGUAGE (WSDL)

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

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

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

Report a bug

6.4. WEB SERVICE STACK

The web service stack is a layer of software. Its role is to make web services available to BPEL processes.

Report a bug

6.5. JAVA API FOR XML WEB SERVICES (JAX-WS)

The Java API for XML Web Services (JAX-WS) is a Java API that allows you to create web services. The

JAX-WS handler mechanism is used by the web service to invoke a user-specified class whenever a message (or fault) is sent or received. The handler is therefore installed in the message pipeline and used to manipulate the message header or body as required.

Report a bug

6.6. JAX-WS HANDLERS AND BPEL

BPEL integrates with JBossWS through the JAX-WS API. This allows BPEL to support both the JBossWS native and Apache CXF web service stacks. Normally you install handlers either programmatically or through a **`HandlerChain`** annotation on the Java interface representing the web service. However, in the case of a process deployed to BPEL, the JAX-WS service is created dynamically upon deployment.

Report a bug

6.7. CONFIGURE A JAX-WS HANDLER FOR BPEL

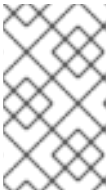
Procedure 6.1. Task

1. Navigate to where you have placed your BPEL process definition and deployment descriptor.
2. Create a file called **`jws_handler.xml`**: **`vi jws_handler.xml`**.
3. Add your configuration settings to the file. Here is an example pertaining to the **`bpel_service_handler`** quickstart:

```
<handler-chains xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee">
    <handler-chain>
  <handler>
   <handler-name>JAXWSHandler</handler-name>
   <handler-
class>org.jboss.soa.bpel.examples.jaxws.JAXWSHandler</handler-class>
   <init-param>
       <param-name>TestParam</param-name>
       <param-value>TestValue</param-value>
   </init-param>
  </handler>
    </handler-chain>
</handler-chains>
```

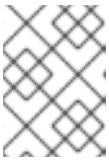
   This file is in the standard JAX-WS handler chain configuration format. One or more handler elements can be specified, with each handler defining a name and a class. (The handler configuration can optionally provide initialization parameters that are passed to the handler implementation's **`init`** method).

4. Save the file and exit.



NOTE

This mechanism only installs JAX-WS handlers on the provider web service. At present, you cannot configure JAX-WS handlers for the client end-points that are used to invoke external web services from a BPEL process.



NOTE

To learn more, study the example of this mechanism that is provided with the **`service_handler`** quick start.

Report a bug

6.8. JWS_HANDLER.XML

`jws_handler.xml` is an XML-based configuration file for the JAX-WS API.

Report a bug

6.9. APACHE CXF

Apache CXF is an open source framework for developing service-oriented architectures (SOAs). CXF helps you build and develop services using frontend programming APIs, like JAX-WS and JAX-RS. These services can use a variety of protocols such as SOAP, XML/HTTP, RESTful HTTP, or CORBA and work over a variety of transports such as HTTP, JMS or JBI.

For more information about CXF, refer to http://cxf.apache.org/docs/.

Report a bug

6.10. CONFIGURE APACHE CXF FOR USE AS A SERVER END-POINT

Prerequisites

- Apache CXF

Procedure 6.2. Task

1. Navigate to the directory containing the BPEL deployment descriptor.
2. Create a file called **`jbossws-cxf.xml`**: **`vi jbossws-cxf.xml`**
3. Add your configuration settings to the file. Here is an example:

```
<beans
  xmlns='http://www.springframework.org/schema/beans'
  xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
  xmlns:beans='http://www.springframework.org/schema/beans'
```

```
  xmlns:jaxws='http://cxf.apache.org/jaxws'
  xsi:schemaLocation='http://cxf.apache.org/core
    http://cxf.apache.org/schemas/core.xsd
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
    http://cxf.apache.org/jaxws
    http://cxf.apache.org/schemas/jaxws.xsd'>

  <bean id="UsernameTokenSign_Request"
     class="org.apache.cxf.ws.security.wss4j.WSS4JInInterceptor">
    <constructor-arg>
      <map>
        <entry key="action" value="UsernameToken Timestamp
Signature"/>
        <entry key="passwordType" value="PasswordDigest"/>
     <entry key="user" value="serverx509v1"/>
        <entry key="passwordCallbackClass"

value="org.jboss.test.ws.jaxws.samples.wsse.ServerUsernamePasswordCa
llback"/>
        <entry key="signaturePropFile"
value="etc/Server_SignVerf.properties"/>
        <entry key="signatureKeyIdentifier"
value="DirectReference"/>
      </map>
    </constructor-arg>
  </bean>

  <bean id="UsernameTokenSign_Response"
     class="org.apache.cxf.ws.security.wss4j.WSS4JOutInterceptor">
    <constructor-arg>
      <map>
        <entry key="action" value="UsernameToken Timestamp
Signature"/>
        <entry key="passwordType" value="PasswordText"/>
     <entry key="user" value="serverx509v1"/>
        <entry key="passwordCallbackClass"

value="org.jboss.test.ws.jaxws.samples.wsse.ServerUsernamePasswordCa
llback"/>
        <entry key="signaturePropFile"
value="etc/Server_Decrypt.properties"/>
        <entry key="signatureKeyIdentifier"
value="DirectReference"/>
        <entry key="signatureParts"
          value="{Element}{http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-
1.0.xsd}Timestamp;{Element}
{http://schemas.xmlsoap.org/soap/envelope/}Body"/>
      </map>
    </constructor-arg>
  </bean>

  <jaxws:endpoint
    id='SecureHelloWorldWS'
```

```
address='http://@jboss.bind.address@:8080/Quickstart_bpel_secure_ser
viceWS'
    implementor='@provider@'>
    <jaxws:inInterceptors>
        <ref bean="UsernameTokenSign_Request"/>
        <bean
class="org.apache.cxf.binding.soap.saaj.SAAJInInterceptor"/>
    </jaxws:inInterceptors>
    <jaxws:outInterceptors>
        <ref bean="UsernameTokenSign_Response"/>
        <bean
class="org.apache.cxf.binding.soap.saaj.SAAJOutInterceptor"/>
    </jaxws:outInterceptors>
  </jaxws:endpoint>


</beans>
```

This example code configures the web service to use the username **token** in conjunction with *digital signature authentication*.

NOTE

The **jaxws:endpoint** element has an attribute called implementor. This attribute defines the Java class implementing the JAX-WS service. The BPEL Console dynamically creates this class automatically. Therefore, it is important that the attribute is set to the value **@provider@** or it will not work.

4. Save the file and exit.

Report a bug

6.11. JBOSSWS-CXF.XML

jbossws-cxf.xml is a user-created XML-based configuration file is a user-created file. It should be placed in a project's **/WEB-INF/** folder alongside the deployment descriptor. Use it to configure Apache CXF as a server end-point. It is also used by jbossws-cxf when deploying a web service based on the use of JAX-WS annotations.

Report a bug

6.12. DEPLOYMENT DESCRIPTOR

A deployment descriptor is an XML-based configuration file for a deployable system artifact. it describes how and where the artifact is to be deployed. You can specify various options and security settings in this file.

Report a bug

6.13. JBOSSWS-CXF-PORTNAME_LOCAL_PART.XML

The various configuration settings for client end-points representing BPEL-invoked web services are currently distributed amongst different files on a per-port basis.

These files adhere to this naming convention: **`jbossws-cxf-{portname_local_part}.xml`**, whereby **`portname_local_part`** represents the local part of the port name of the web service being invoked.

Report a bug

6.14. EXAMPLE OF WSDL FOR AN APACHE CXF CLIENT END-POINT

```
<definitions name='SecureHelloWorldWSService'
  targetNamespace='http://secure_invoke/helloworld' .... >
    <portType name='SecureHelloWorld'>
        ...
    </portType>
    <service name='SecureHelloWorldWSService'>
       <port name='SecureHelloWorldPort' ... >
           ...
       </port>
    </service>
</definitions>
```

Report a bug

6.15. CONFIGURE APACHE CXF FOR USE AS A CLIENT END-POINT

Prerequisites

- Apache CXF

Procedure 6.3. Task

1. To edit the sample **`jbossws-cxf-SecureHelloWorldPort.xml`** CXF configuration file, issue this command: **`vi SOA_ROOT/jboss-as/samples/quickstarts/bpel_secure_invoke/bpelContent/jbossws-cxf-SecureHelloWorldPort.xml`**

2. The configuration information contained within this file is for the CXF bus. Edit the file like this:

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:cxf="http://cxf.apache.org/core"
       xmlns:wsa="http://cxf.apache.org/ws/addressing"

xmlns:http="http://cxf.apache.org/transports/http/configuration"
       xmlns:wsrm-
policy="http://schemas.xmlsoap.org/ws/2005/02/rm/policy"
       xmlns:wsrm-mgr="http://cxf.apache.org/ws/rm/manager"
```

```
    xmlns:beans='http://www.springframework.org/schema/beans'
    xmlns:jaxws='http://cxf.apache.org/jaxws'
    xmlns:ns1='http://secure_invoke/helloworld'
       xsi:schemaLocation="
       http://cxf.apache.org/core
http://cxf.apache.org/schemas/core.xsd
       http://cxf.apache.org/transports/http/configuration
http://cxf.apache.org/schemas/configuration/http-conf.xsd
       http://schemas.xmlsoap.org/ws/2005/02/rm/policy
http://schemas.xmlsoap.org/ws/2005/02/rm/wsrm-policy.xsd
       http://cxf.apache.org/ws/rm/manager
http://cxf.apache.org/schemas/configuration/wsrm-manager.xsd
       http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
 http://cxf.apache.org/jaxws
http://cxf.apache.org/schemas/jaxws.xsd">

  <bean id="UsernameTokenSign_Request"
     class="org.apache.cxf.ws.security.wss4j.WSS4JOutInterceptor" >
    <constructor-arg>
      <map>
        <entry key="action" value="UsernameToken Timestamp
Signature"/>
        <entry key="passwordType" value="PasswordDigest"/>
  <entry key="user" value="clientx509v1"/>
        <entry key="passwordCallbackClass"

value="org.jboss.test.ws.jaxws.samples.wsse.ClientUsernamePasswordCa
llback"/>
        <entry key="signaturePropFile"
value="etc/Client_Sign.properties"/>
        <entry key="signatureKeyIdentifier"
value="DirectReference"/>
        <entry key="signatureParts"
          value="{Element}{http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-
1.0.xsd}Timestamp;{Element}
{http://schemas.xmlsoap.org/soap/envelope/}Body"/>
      </map>
    </constructor-arg>
  </bean>

  <bean id="UsernameTokenSign_Response"
     class="org.apache.cxf.ws.security.wss4j.WSS4JInInterceptor" >
    <constructor-arg>
      <map>
        <entry key="action" value="UsernameToken Timestamp
Signature"/>
        <entry key="passwordType" value="PasswordText"/>
     <entry key="user" value="serverx509v1"/>
        <entry key="passwordCallbackClass"

value="org.jboss.test.ws.jaxws.samples.wsse.ClientUsernamePasswordCa
llback"/>
        <entry key="signaturePropFile"
value="etc/Client_Encrypt.properties"/>
```

```
        <entry key="signatureKeyIdentifier"
value="DirectReference"/>
      </map>
    </constructor-arg>
  </bean>

  <cxf:bus>
    <cxf:outInterceptors>
        <ref bean="UsernameTokenSign_Request"/>
        <bean
class="org.apache.cxf.binding.soap.saaj.SAAJOutInterceptor"/>
    </cxf:outInterceptors>
    <cxf:inInterceptors>
        <ref bean="UsernameTokenSign_Response"/>
        <bean
class="org.apache.cxf.binding.soap.saaj.SAAJInInterceptor"/>
    </cxf:inInterceptors>
  </cxf:bus>

</beans>
```

These settings configure the web service client to utilise the username token and digital signature authentication.

3. Save the file and exit.

Report a bug

6.16. BPEL PROCESSES AND WEB SERVICES

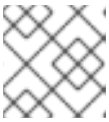
When you deploy a BPEL process, a web service is automatically created. This web service represents the service end-point and it is based on the WSDL description that is bundled with the process that is being deployed.

Report a bug

6.17. DISPLAY A WSDL

Procedure 6.4. Task

1. Look up the end-point URL of the service which has the "?wsdl" suffix.
2. Once it is displayed, the <soap:address> will, by default, become associated with the server's "host bind" address.



NOTE

This address is defined in the ${jboss.bind.address} property.

Report a bug

6.18. CONFIGURE A BPEL CLIENT END-POINT TO INVOKE A SECURE EXTERNAL WEB SERVICE

Procedure 6.5. Task

1. Launch your test editor and create a new file.
2. Add the relative locations of the key-store and trust-store files.
3. Save the file as jboss-wsse-client.xml in the top level directory alongside your BPEL process and exit.

NOTE

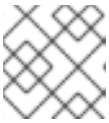
For more information, see the secure_invoke_native quickstart.

Report a bug

6.19. CONFIGURE A BPEL SERVER END-POINT TO PROVIDE A SECURE WEB SERVICE

Procedure 6.6. Task

1. Launch your test editor and create a new file.
2. Add the relative locations of the key-store and trust-store files.
3. Save the file as jboss-wsse-server.xml in the top level directory alongside your BPEL process and exit.



NOTE

For more information, see the secure_service_native quickstart.

Report a bug

CHAPTER 7. ADVANCED BPEL ENGINE INTEGRATION

7.1. BI-DIRECTIONAL LOOSE COUPLING

Bi-directional loose coupling is provided through web services. For example, an ESB action may invoke a BPEL process via a web service. This web service would be represented by a WSDL interface. Likewise, a BPEL process can invoke an ESB-managed service that can present itself as a web service.

Report a bug

7.2. ENTERPRISE SERVICE BUS

An enterprise service bus is a concrete implementation of an abstract SOA design philosophy. An enterprise service bus (ESB) has two roles: it provides message routing functionality and allows you to register services. The enterprise service bus that lies at the center of the JBoss Enterprise SOA Platform is called JBoss ESB.

An enterprise service bus deals with infrastructure logic, not business logic (which is left to higher levels). Data is passed through the enterprise service bus on its way between two or more systems. Message queuing may or may not be involved. The ESB can also pass the data to a transformation engine before passing it to its destination.

Report a bug

7.3. USE THE BPELINVOKE ESB ACTION

Prerequisites

- The BPEL Engine must be installed in the same Java virtual machine as the ESB
- The requested process must have been deployed to the local BPEL Engine.

Procedure 7.1. Task

1. Open a **`jboss-esb.xml`** file in your text editor.
2. Add the the BPELInvoke ESB action.

   The following example shows the action being used together with the **`bpel_esb_helloworld`** quickstart:

```
      <action name="action2"
class="org.jboss.soa.esb.actions.bpel.BPELInvoke">
    <property name="service" value="
{http://www.jboss.org/bpel/examples/wsdl}HelloService"/>
    <property name="port" value="HelloPort" />
    <property name="operation" value="hello" />
    <property name="requestPartName" value="TestPart" />
    <property name="responsePartName" value="TestPart" />
</action>
```

3. Save the file and exit.

Report a bug

7.4. BPELINVOKE ESB ACTION CLASS

The BPELInvoke ESB action class (`org.jboss.soa.esb.actions.bpel.BPELInvoke`) can be used to call upon the BPEL Engine from within ESB services.

Report a bug

7.5. BPELINVOKE ESB ACTION CLASS PROPERTIES

Table 7.1. Properties

Property	Explanation
service	This property is mandatory. It defines the service name registered in the WSDL which is associated with the deployed BPEL process.
port	This property is optional. It defines the name of the port associated with the deployed BPEL process that is registered in the WSDL. This parameter is only required if port-specific end-point configuration information has been registered as part of the BPEL process deployment.
operation	This property is mandatory. It represents the WSDL operation being invoked.
requestPartName	Use this optional property to define the WSDL message part to which the inbound ESB message content should be mapped. Only use this property where the ESB message does not already represent a multi-part message.
responsePartName	Use this optional property to extract the content of a response multi-part WSDL message and place it in the ESB message being passed to the next action in the pipeline. (If this property is not defined, the complete multi-part message value will be placed in the ESB message.)
abortOnFault	Use this optional property to indicate whether a fault generated during the invocation of a BPEL process should be treated as a message (when the value of this property is false) or as an exception that will abort the ESB service. The default value is true, which aborts the service.

This ESB action supports inbound messages. The content of these messages must be defined as one of the following items:

Table 7.2. Properties

Item	Explanation

Item	Explanation
DOM	If the message content is a DOM document or element, it can be used as the complete multi-part message or as the content of a message part. Use the requestPartName property to define it.
Java String	If the message content is a string representation of an XML document, the requestPartName is optional. If not specified, the document must represent the multi-part message. If the message content is a string that does not represent an XML document, the requestPartName must be specified.

When the message content represents every single part of a *multi-part message*, it must be defined as a top-level element (which you can name anything). Its child elements must come immediately below it as these represent each part of the message. Each of these child elements must in turn have their own single child element. This element represents the value of the named part.

This is the structure of a multi-part message:

```
<message>
 <TestPart>
  Hello World
 </TestPart>
</message>
```

The top element (the message tag) is unimportant. The elements at the next level represent the part names: in this case there is only one, namely **`TestPart`**. This part is a text node, in this case called **`Hello World`**. (However it could also have been an element representing the root node of a more complex XML value.)

Report a bug

7.6. BPELINVOKE AND THE ACTION PIPELINE

When a WSDL operation is run, a response message will normally be returned from the BPELInvoke action. This message is then placed in the action pipeline ready for processing by the next action. (If no further actions have been defined, it is returned to the service client.)

Report a bug

7.7. ACTION PIPELINE

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

The action pipeline works in two stages: normal processing followed by outcome processing. In the first stage, the pipeline calls the process method(s) on each action (by default it is called "process") in

sequence until the end of the pipeline has been reached or an error occurs. At this point the pipeline reverses (the second stage) and calls the outcome method on each preceding action (by default it is processException or processSuccess). It starts with the current action (the final one on success or the one which raised the exception) and travels backwards until it has reached the start of the pipeline.

Report a bug

CHAPTER 8. BPEL AND REST

8.1. BPEL CONSOLE RESTFUL SERVICES

This is a list of Restful services that are used by the BPEL Console.

Table 8.1. BPEL Console RESTful Services

Method	Path	Description	Consumes	Produces
Server Info (General REST server information)	GET	/gwt-console-server/rs/server/status	*/*	application/json
-	GET	/gwt-console-server/rs/server/resources/{project}	*/*	text/html
Process Management(Process related data.)	GET	/gwt-console-server/rs/process/definitions	*/*	application/json
-	GET	/gwt-console-server/rs/process/definition/{id}/instances	*/*	application/json
-	GET	/gwt-console-server/rs/process/instance/{id}/dataset	*/*	text/xml
-	POST	/gwt-console-server/rs/process/instance/{id}/end/{result}	*/*	application/json
-	GET	/gwt-console-server/rs/process/definition/{id}/image	*/*	image/*
-	GET	/gwt-console-server/rs/process/definition/{id}/image/{instance}	*/*	image/*
Process Engine(Process runtime state)	GET	/gwt-console-server/rs/engine/deployments	*/*	application/json

Method	Path	Description	Consumes	Produces
Process History(Process History Service)	GET	/gwt-console-server/rs//history/definition/{id}/instances	*/*	applications/json
-	GET	/gwt-console-server/rs//history/definitions	*/*	application/json
-	GET	/gwt-console-server/rs//history/definition/{id}/instances	*/*	application/json
-	GET	/gwt-console-server/rs//history/instance/{id}/activities	*/*	application/json
-	GET	/gwt-console-server/rs//history/instance/{id}/events	*/*	application/json
-	GET	/gwt-console-server/rs//history/definition/{id}/instances/completed	*/*	application/json
-	GET	/gwt-console-server/rs//history/definition/{id}/instances/failed	*/*	application/json
-	GET	/gwt-console-server/rs//history/definition/{id}/instances/terminated	*/*	application/json
-	GET	/gwt-console-server/rs//history/definition/{id}/instances/chart/completed	*/*	application/json
-	GET	/gwt-console-server/rs//history/definition/{id}/instances/chart/failed	*/*	application/json

Report a bug

CHAPTER 9. FAULT HANDLING

9.1. BPELINVOKE FAULT HANDLING

Depending on how you choose to configure the system, you can receive the fault as an ESB message or allow it to cause an exception which aborts the action pipeline. The configuration property that determines which behavior is to be used used is called abortOnFault. The default value for this property is "true".

Report a bug

9.2. BPELINVOKE FAULT HANDLING REFERENCE

The following example material uses the **Loan Fault** quickstart:

```
        <action name="action2"
class="org.jboss.soa.esb.actions.bpel.BPELInvoke">
  <property name="service" value="{http://example.com/loan-
approval/wsdl/}loanService"/>
 <property name="operation" value="request" />
 <property name="abortOnFault" value="true" />
</action>
```

A WSDL fault reports two pieces of information: the fault type and the details of the fault. Each of these is returned in a separate part of message's body:

1. Fault code: **javax.xml.namespace.QName**

   ESB message body part: **org.jboss.soa.esb.message.fault.detail.code**

   This body part identifies the specific WSDL fault returned by the BPEL process.

   WARNING

   The version of the *QName* used by the **JBoss Enterprise SOA Platform** server is found in the **lib/endorsed/stax-api.jar** file. (If this file is absent, you will encounter a class version exception.)

2. Fault code (as a textual representation of the QName).

   ESB message body part: **org.jboss.soa.bpel.message.fault.detail.code**

   This body part returns the textual representation of the fault code's QName. This representation is of the form **{namespace}localpart**. (To convert it back into a QName, use the **javax.xml.namespace.QName.valueOf(String)** method.)

3. Fault details

ESB message body part: **`org.jboss.soa.esb.message.fault.detail.detail`**

This body part contains the textual representation of the message content associated with the fault.

Report a bug

CHAPTER 10. SAML SUPPORT

10.1. SECURITY ASSERTION MARKUP LANGUAGE (SAML)

Security Assertion Markup Language (SAML) is an XML-based OASIS standard method for exchanging security data between an identity provider and a service or consumer.

Report a bug

10.2. PICKETLINK

PicketLink is an umbrella project covering with a number of security and identity management sub-projects.

Report a bug

10.3. SAML TOKEN

A SAML token is designed to pass information about an end user between an identity provider and a web service.

Report a bug

10.4. SAML SUPPORT IN BPEL

If the ESB service uses PicketLink to obtain a SAML token, this assertion can be passed to the invoked BPEL process by means of the requestSAMLPartName property:

```
        <action name="action2"
class="org.jboss.soa.esb.actions.bpel.BPELInvoke">
 <property name="service" value="
{http://simple_invoke/helloworld}HelloHeaderWSService"/>
 <property name="operation" value="sayHi" />
 <property name="requestPartName" value="sayHello" />
 <property name="responsePartName" value="sayHelloResponse" />
 <property name="requestSAMLPartName" value="Security" />
</action>
```

The **requestSAMLPartName** identifies the name of a message part. You must define this part as a WS-Security element:

```
<part name="Security" element="wsse:Security"
xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-secext-1.0.xsd" />
```

Report a bug

APPENDIX A. REVISION HISTORY

Revision 5.3.1-93.400 **2013-10-31** **Rüdiger Landmann**

Rebuild with publican 4.0.0

Revision 5.3.1-93 **Tue Feb 05 2013** **David Le Sage**

Built from Content Specification: 6892, Revision: 371689 by dlesage