RiftSaw 2.0-CR1

User Guide

by Gary Brown, Kurt Stam, and Heiko Braun

1. Introduction 1
  1.1. Overview 1
2. Administration 2
  2.1. Overview 2
  2.2. BPM Console 2
    2.2.1. Overview 2
    2.2.2. Logging in 2
    2.2.3. Deployed Process Definitions 3
    2.2.4. Process Instances 3
3. Deploying BPEL Processes 6
  3.1. Overview 6
  3.2. Direct deployment to JBossAS server 6
  3.3. Eclipse based Deployment 7
4. JBoss ESB Integration 13
  4.1. Overview 13
  4.2. Using the *BPELInvoke* ESB action 13
    4.2.1. Fault Handling 15

Introduction

1.1. Overview

This is the User Guide for the RiftSaw BPEL process engine.

RiftSaw provides a JBoss AS integration for the Apache ODE BPEL engine. For detailed information on executing BPEL processes within Apache ODE, we would refer the reader to the Apache ODE website and documentation.

In addition to the ability to run the Apache ODE engine within JBoss AS, the RiftSaw project also provides a GWT based administration console, replaces the Axis2 based transport with JBossWS (which can be configured to use Apache CXF), and provides tighter integration with JBossESB.

Administration

2.1. Overview

This section describes the administration capabilities associated with RiftSaw.

2.2. BPM Console

2.2.1. Overview

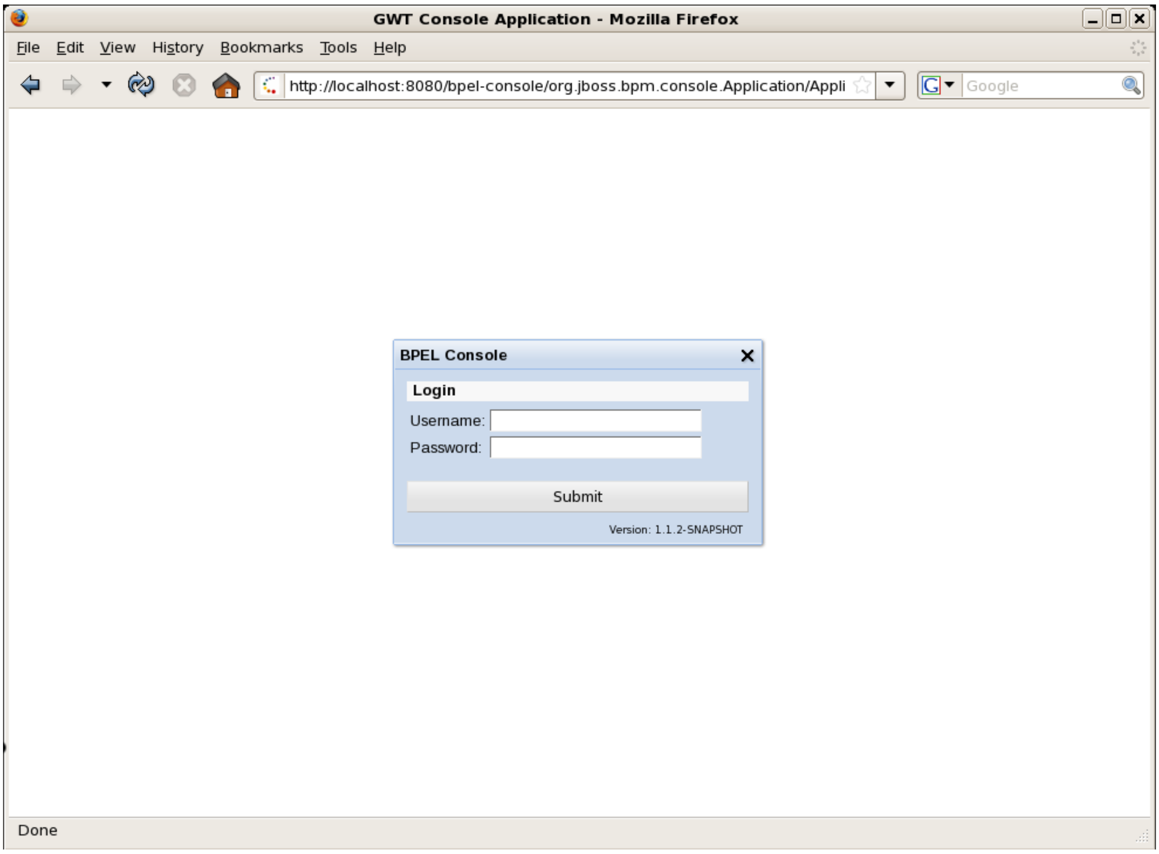
This section provides an overview of the BPEL Console. The console provides the ability to view:

- The process definitions deployed to the BPEL engine
- The process instances executing in the BPEL engine

2.2.2. Logging in

The BPEL console can be located using the URL: http://localhost:8080/bpel-console.

The first screen that is presented is the login screen:


GWT Console Application - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/bpel-console/org.jboss.bpm.console.Application/Appli
Google
BPEL Console
Login
Username:
Password:
Submit
Version: 1.1.2-SNAPSHOT
Done

The default username is *admin* with password *password*.

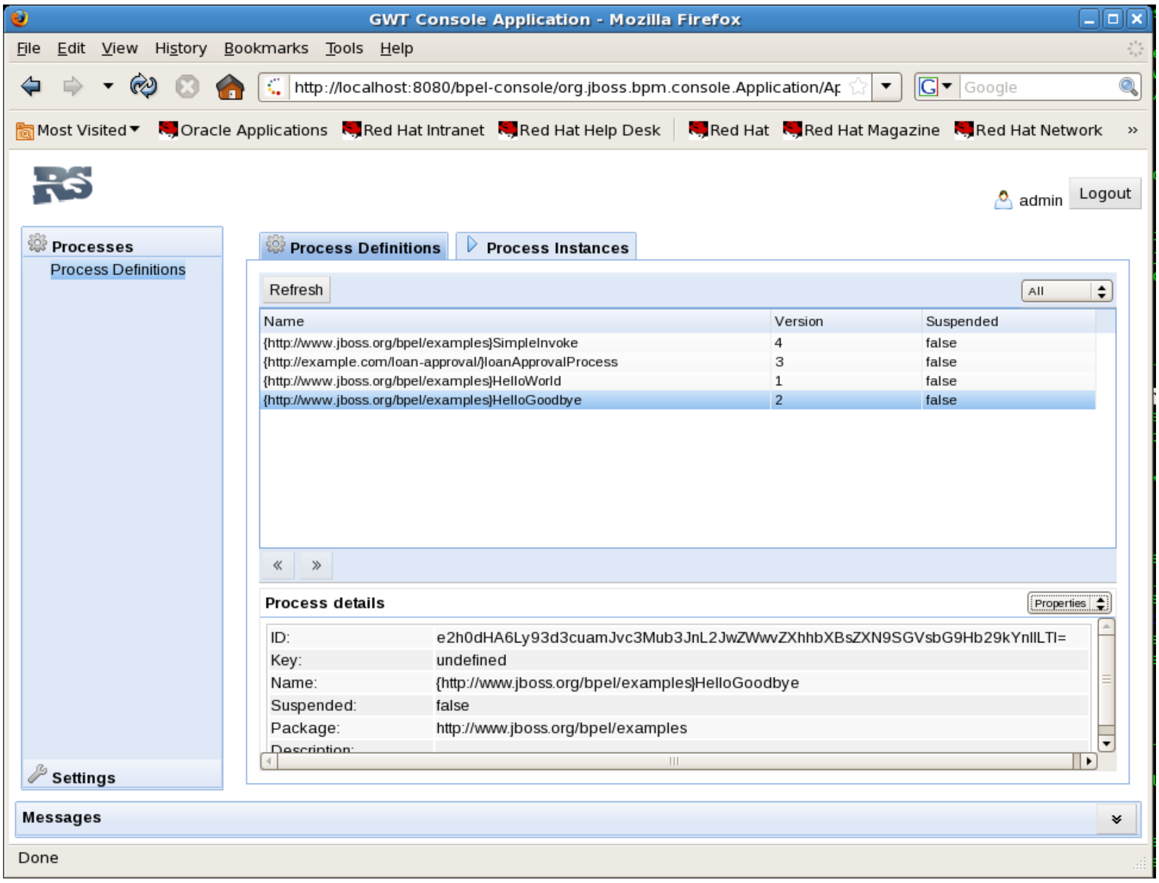
The Access Control mechanism used by the admin console is configured in the `$deployFolder/bpel-console/bpel-identity.sar/META-INF/jboss-service.xml`. The JAAS login module is initially set to use a property file based access mechanism, but can be replaced to use any appropriate alternative implementation.

The users for the default mechanism are configured in the property file `$deployFolder/bpel-console/bpel-identity.sar/bpel-users.properties`. The entries in this file represent *username=password*.

The user roles for the default mechanism are configured in the property file `$deployFolder/bpel-console/bpel-identity.sar/bpel-roles.properties`. The entries in this file represent *username=role*. The only role of interest currently is *administrator*.

2.2.3. Deployed Process Definitions

Once logged in, the 'Process Definitions' tab shows the currently deployed BPEL processes.



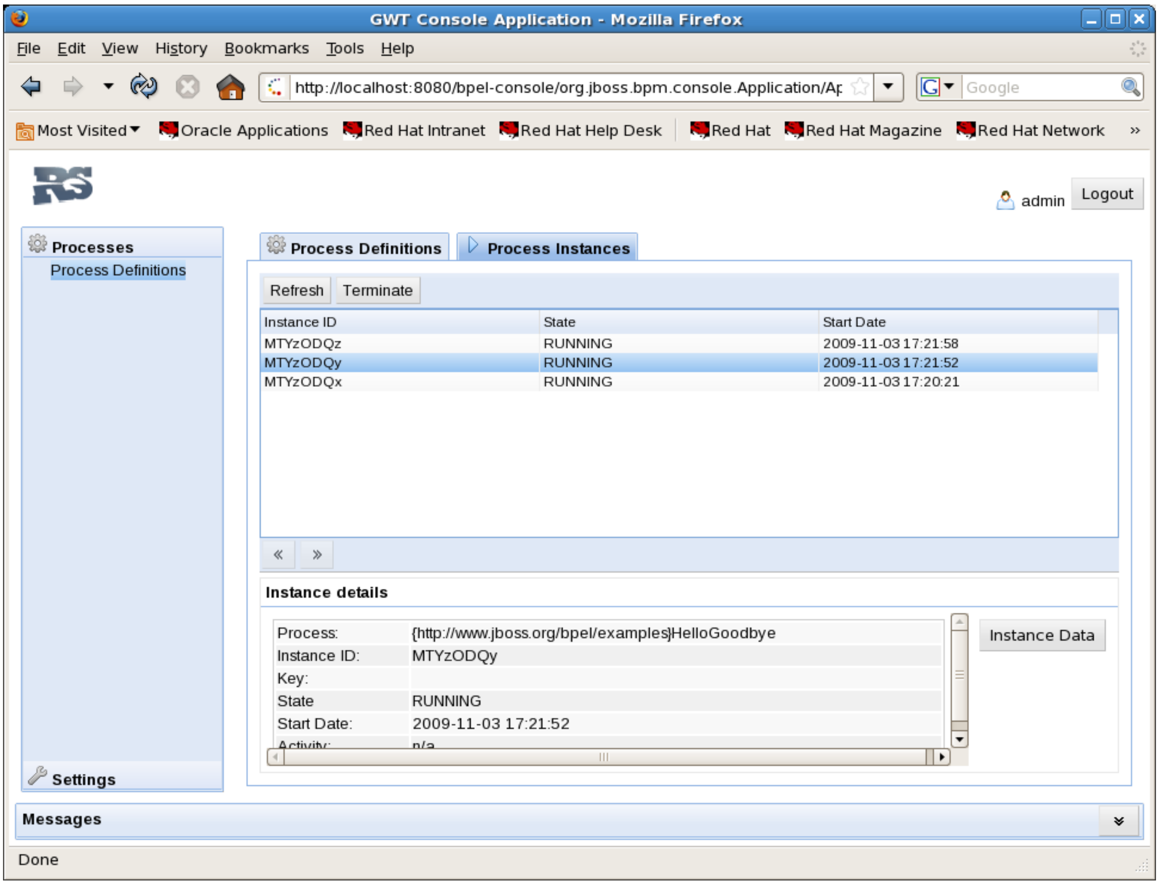
Name	Version	Suspended
{http://www.jboss.org/bpel/examples}SimpleInvoke	4	false
{http://example.com/loan-approval/}loanApprovalProcess	3	false
{http://www.jboss.org/bpel/examples}HelloWorld	1	false
{http://www.jboss.org/bpel/examples}HelloGoodbye	2	false

ID:	e2h0dHA6Ly93d3cuamJvc3Mub3JnL2JwZWwvZXhhbXBsZXN9SGVsbG9Hb29kYnllLTI=
Key:	undefined
Name:	{http://www.jboss.org/bpel/examples}HelloGoodbye
Suspended:	false
Package:	http://www.jboss.org/bpel/examples
Description:	

When a process definition is selected, the details will be displayed in the lower panel.

2.2.4. Process Instances

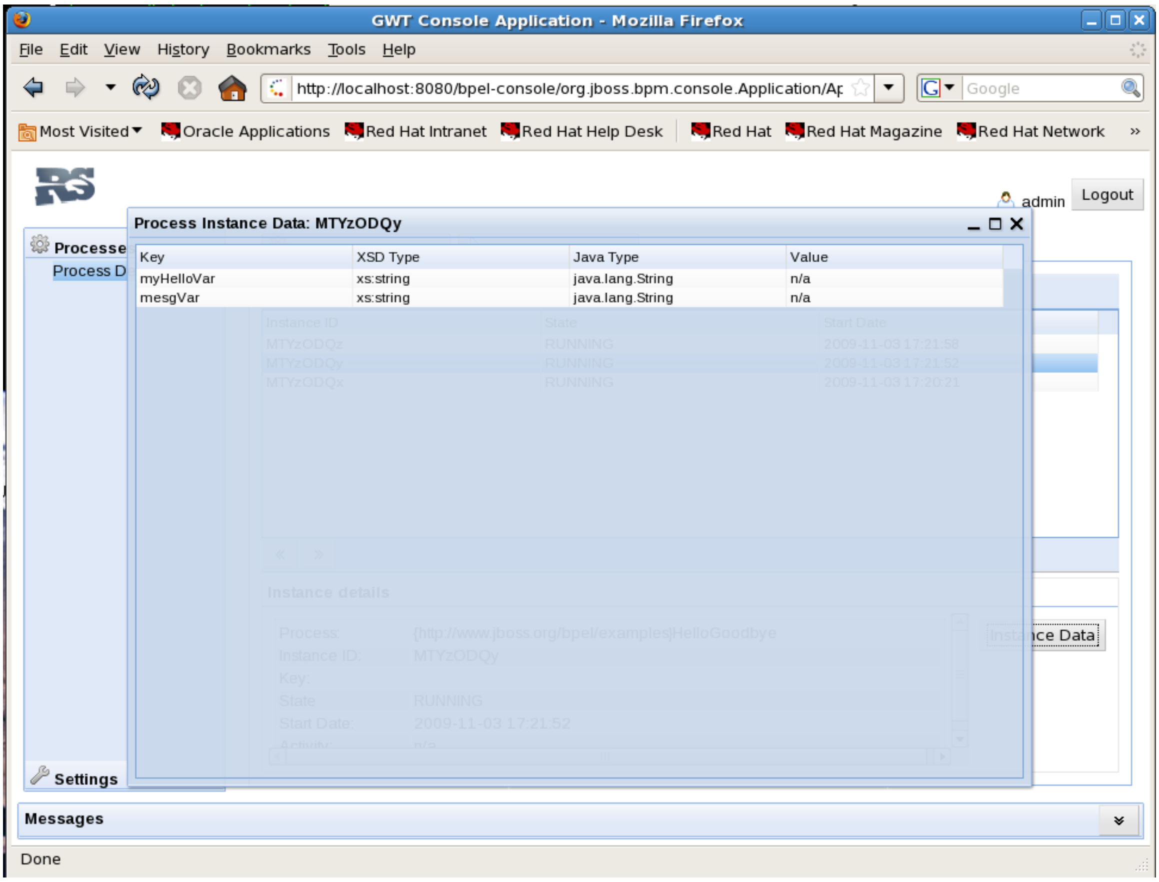
The 'Process Instances' tab shows the currently executing BPEL process instances. Before selecting this tab, you must choose a process definition.



Instance ID	State	Start Date
MTYzODQz	RUNNING	2009-11-03 17:21:58
MTYzODQy	RUNNING	2009-11-03 17:21:52
MTYzODQx	RUNNING	2009-11-03 17:20:21

Process:	{http://www.jboss.org/bpel/examples}HelloGoodbye
Instance ID:	MTYzODQy
Key:	
State	RUNNING
Start Date:	2009-11-03 17:21:52
Activity:	n/a

When a process instance has been selected, its details will be displayed in the lower properties window. The *Instance Data* button will also become enable, allowing further detail about the process to be displayed.



Process Instance Data: MTYzODQy

Key	XSD Type	Java Type	Value
myHelloVar	xs:string	java.lang.String	n/a
mesgVar	xs:string	java.lang.String	n/a

Instance ID	State	Start Date
MTYzODQz	RUNNING	2009-11-03 17:21:58
MTYzODQy	RUNNING	2009-11-03 17:21:52
MTYzODQx	RUNNING	2009-11-03 17:20:21

Instance details

Process:	{http://www.jboss.org/bpel/examples}HelloGoodbye
Instance ID:	MTYzODQy
Key:	
State	RUNNING
Start Date:	2009-11-03 17:21:52
Activity:	n/a

Deploying BPEL Processes

3.1. Overview

This section outlines the mechanisms that can be used to deploy a BPEL process to RiftSaw BPEL engine running within a JBoss AS server.

3.2. Direct deployment to JBossAS server

The direct deployment approach is demonstrated using an *Ant* script in each of the quickstart examples. For 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>
```

This excerpt from the *Ant* build file for the *hello_world* quickstart example shows that deploying a RiftSaw BPEL process using *Ant* is very straightforward. The main points of interest are:

- It is necessary to identify the location of the JBoss AS server in which the BPEL process will be deployed. This is achieved in this example by referring to the `deployment.properties` file that has been configured in the RiftSaw distribution (install folder).

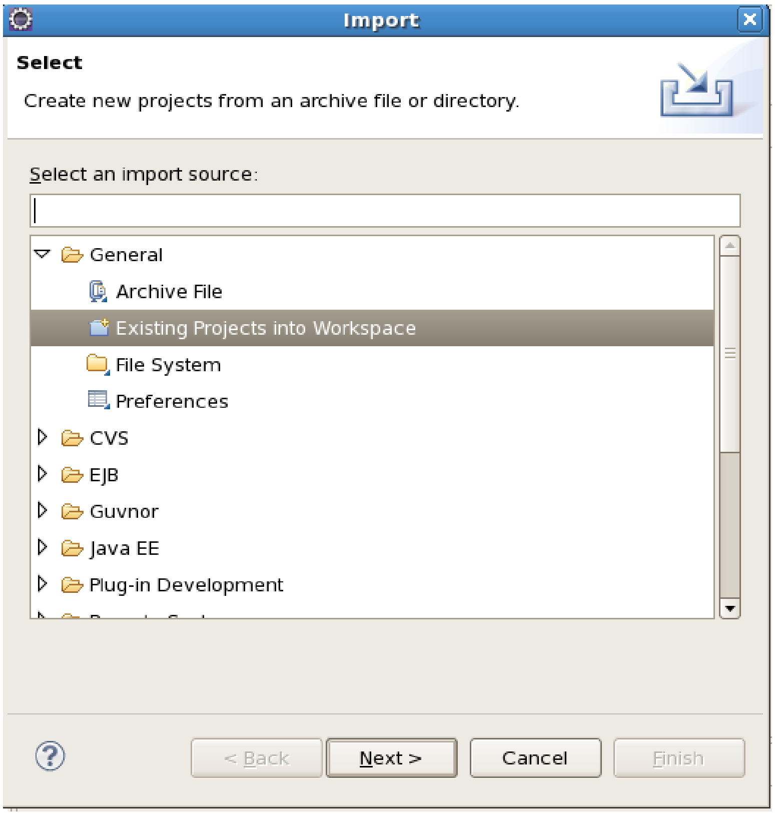
- If a versioned approach is being used, so that multiple versions of the same BPEL process may be deployed at one time, then the name of the archive (jar) containing the BPEL process (and associated artifacts) has a version number suffix. This would need to be manually incremented for each distinct version of the BPEL process being deployed.

- The next step is to define the *deploy* target, which will create the BPEL process archive, using the contents of the bpel sub-folder in this case, and store it within the JBoss AS server's `deploy` folder.
- The final step is to define the *undeploy* target, which simply removes the BPEL process archive from the JBoss AS server's `deploy` folder.

3.3. Eclipse based Deployment

This section will explain how to deploy an Eclipse BPEL project to the RiftSaw BPEL engine running in a JBossAS server.

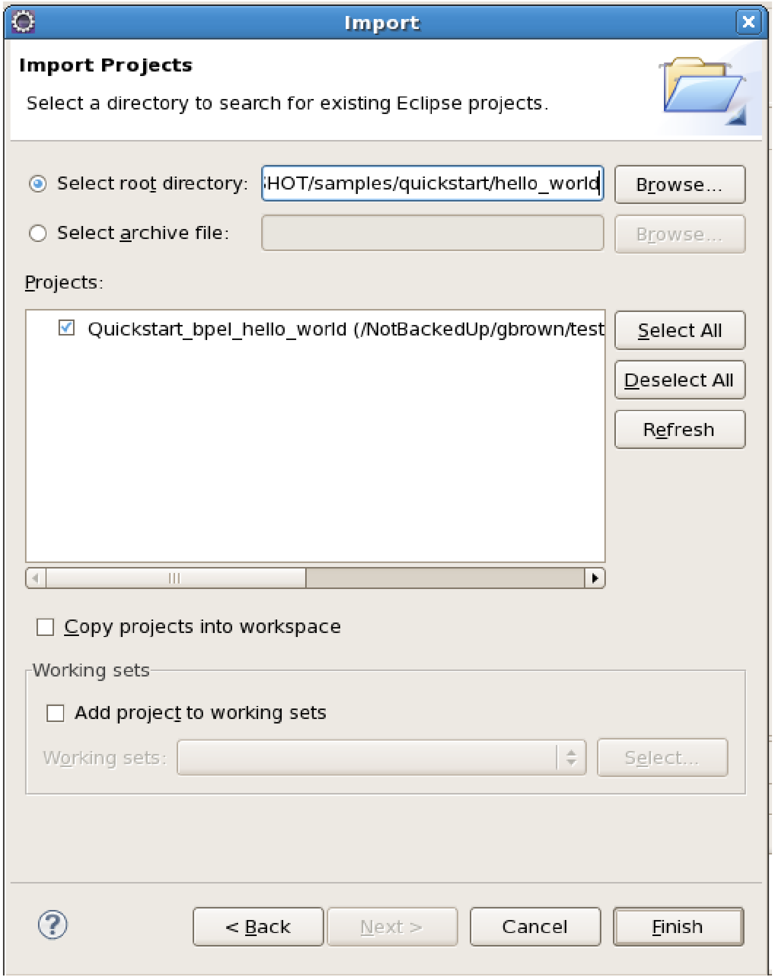
The first step is to create or import the Eclipse BPEL project. In this case we are going to import an existing project from the `${RiftSaw}/samples/quickstart/hello_world` folder. This can be achieved by selecting the *Import* ... menu item associated with the lefthand navigator panel in Eclipse, and then select the *General->Existing Projects into Workspace* entry and press the *Next* button.



Select

Create new projects from an archive file or directory.

Then press the *Browse* button and navigate to the `hello_world` quickstart folder. Once located, press the *Finish* button.



Import

Import Projects

Select a directory to search for existing Eclipse projects.

● Select root directory: HOT/samples/quickstart/hello_world

○ Select archive file:

Projects:

☑ Quickstart_bpel_hello_world (/NotBackedUp/gbrown/test

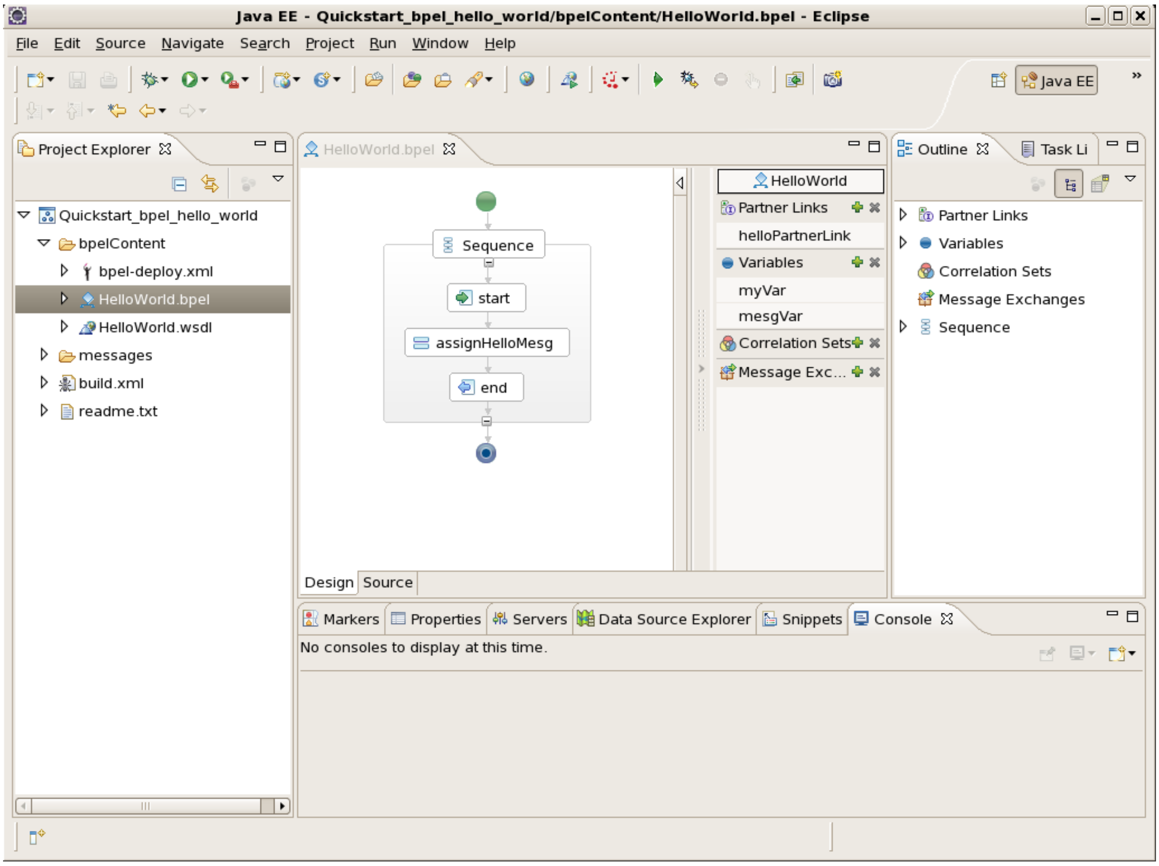
☐ Copy projects into workspace

Working sets

☐ Add project to working sets

Working sets:

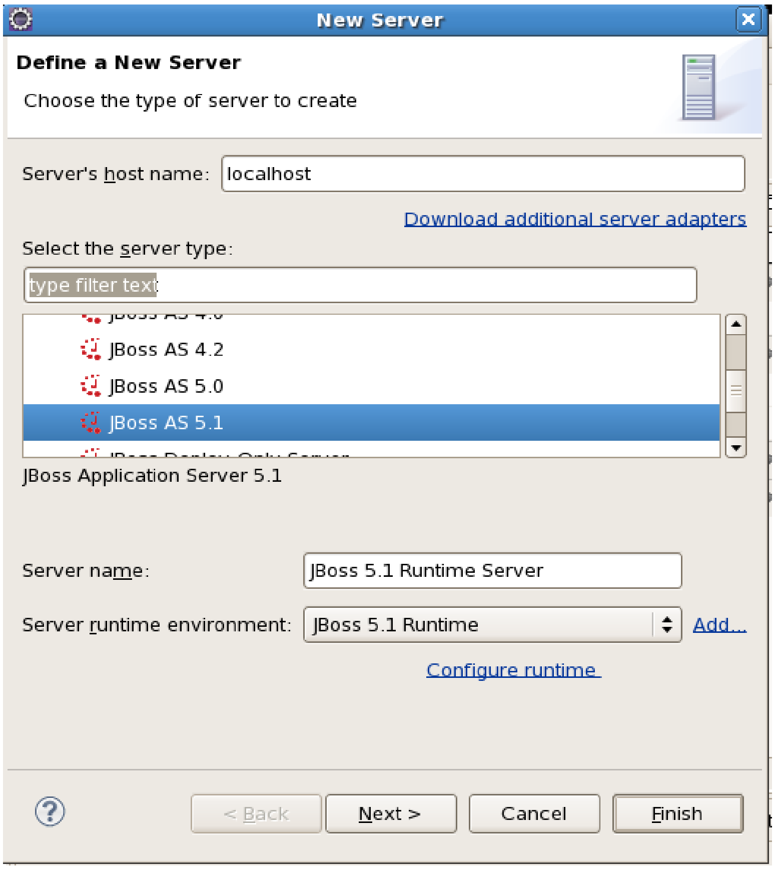
Once the project has been imported, you can inspect the contents, such as the BPEL process and WSDL description.



No consoles to display at this time.

The next step is to create a server configuration for the JBoss AS environment in which the RiftSaw BPEL engine has previously been installed. From the Eclipse *Java EE* perspective, the *Server* tab should be visible in the lower region of the Eclipse window. If this view is not present, then go to the *Window->Show Views->Servers* menu item to open the view explicitly.

In the *Servers* view, right click and select the *New->Server* menu item.



New Server

Define a New Server

Choose the type of server to create

Server's host name: localhost

Select the server type:

JBoss AS 4.2

JBoss AS 5.0

JBoss AS 5.1

JBoss Application Server 5.1

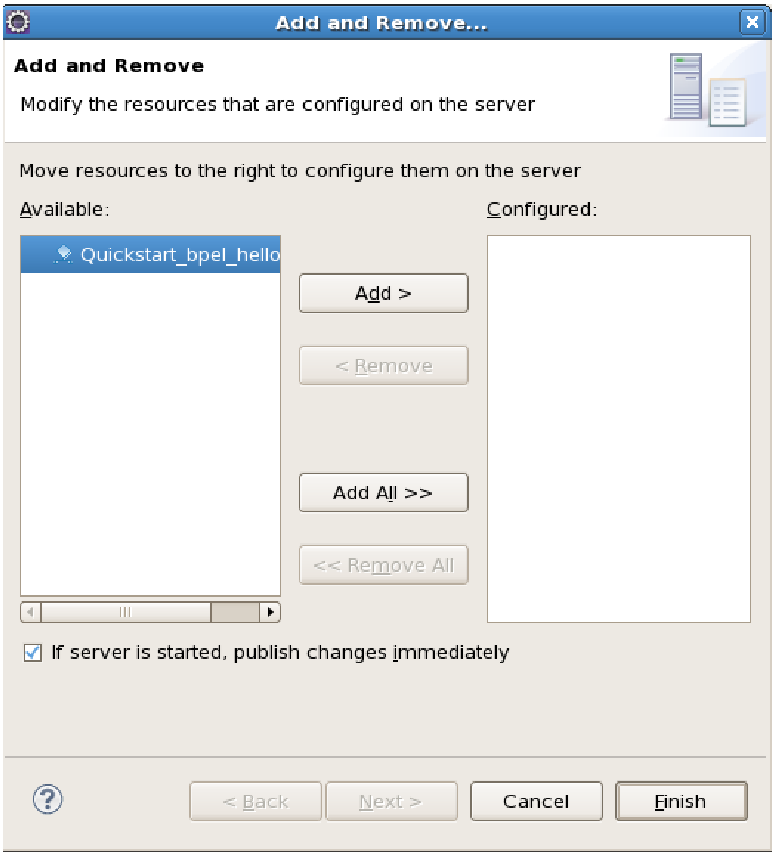
Server name: JBoss 5.1 Runtime Server

Server runtime environment: JBoss 5.1 Runtime

Select the appropriate JBoss AS version, and then press *Finish*.

Before being able to deploy an example, we should start the new server. This can be achieved by right clicking on the server in the *Servers* tab, and selecting the *Start* menu item. The output from the server will be displayed in the *Console* tab.

Once the server has been started, right click on the server entry again, and select the *Add and Remove ...* menu item.



Add and Remove

Modify the resources that are configured on the server

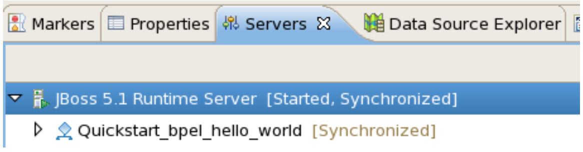
Move resources to the right to configure them on the server

Available: Quickstart_bpel_hello

Configured:

☑ If server is started, publish changes immediately

Select the *Quickstart_bpel_hello_world* project, press the *Add* button and the press the *Finish* button. This will cause the project to be deployed to the server.

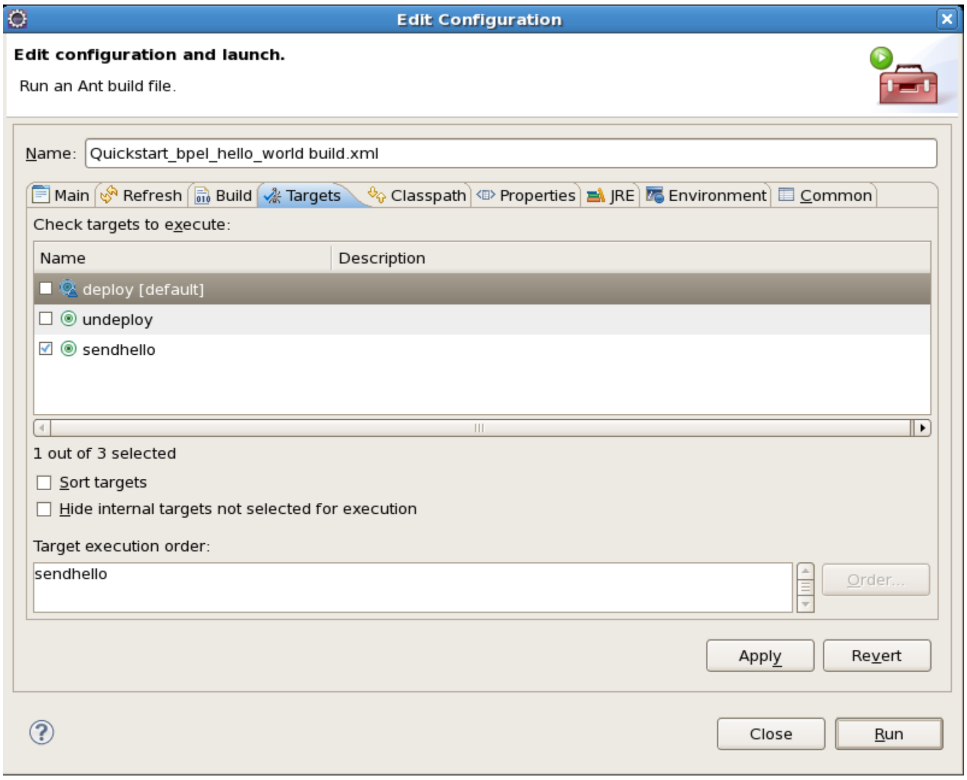


JBoss 5.1 Runtime Server [Started, Synchronized]

Quickstart_bpel_hello_world [Synchronized]

Once the project has been deployed, it will show up as an entry below the server in the *Servers* tab.

The final step is to test the deployed BPEL process. In this example, we can do this using the *ant* script provided with the quickstart sample. Right click on the `build.xml` file in the root folder of the project, and select the *Run As->Ant Build* ... menu item. NOTE: It is important to select the menu item with the *"..."*, as this provides a dialog window to enable you to select which *ant target* you wish to perform.



Edit configuration and launch.

Run an Ant build file.

Name: Quickstart_bpel_hello_world build.xml

Check targets to execute:

☐ deploy [default]

☐ undeploy

☑ sendhello

1 out of 3 selected

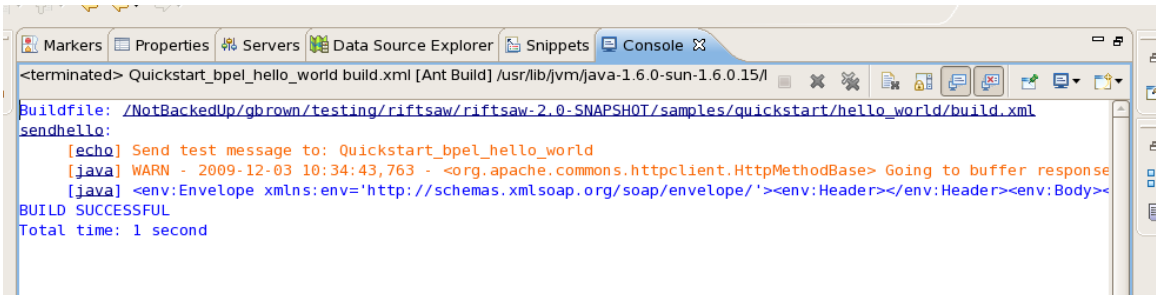
☐ Sort targets

☐ Hide internal targets not selected for execution

Target execution order:

sendhello

Deselect the *deploy* target, and select the *sendhello* target, before pressing the *Run* button. This was send a test 'hello' message to the server, and then display the response in the *Console* tab.



```
<terminated> Quickstart_bpel_hello_world build.xml [Ant Build] /usr/lib/jvm/java-1.6.0-sun-1.6.0.15/l
Buildfile: /NotBackedUp/gbrown/testing/riftsaw/riftsaw-2.0-SNAPSHOT/samples/quickstart/hello_world/build.xml
sendhello:
     [echo] Send test message to: Quickstart_bpel_hello_world
     [java] WARN - 2009-12-03 10:34:43,763 - <org.apache.commons.httpclient.HttpMethodBase> Going to buffer response
     [java] <env:Envelope xmlns:env='http://schemas.xmlsoap.org/soap/envelope/'><env:Header></env:Header><env:Body><
BUILD SUCCESSFUL
Total time: 1 second
```

You can then use the menu associated with the project, contained in the server, to undeploy the project (using the *Add and Remove ...* menu item) and finally use the menu associated with the server itself to *Stop* the server.

JBoss ESB Integration

4.1. Overview

This section outlines the support provided for the direct integration between RiftSaw and JBossESB.

Bi-directional loose integration is available through the use of web services. For example, an ESB action may invoke a BPEL process running within RiftSaw by invoking the appropriate web service represented by a WSDL interface. Similarly, a BPEL process can invoke an ESB managed service that is capable of presenting itself as a web service.

However this section will describe how integration between RiftSaw and JBossESB actions can be achieved without the use of web services (i.e. WSDL and SOAP).

4.2. Using the *BPELInvoke* ESB action

The *BPELInvoke* ESB action can be used within a *jboss-esb.xml* to request an invocation on a BPEL process running inside RiftSaw. The only constraints are that RiftSaw is installed within the same Java VM and that the requested BPEL process must have been deployed to the local RiftSaw engine.

The following example illustrates the *BPELInvoke* ESB action being used as part of the *bpel_helloworld* sample.

```
<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="operation" value="hello" />
    <property name="requestPartName" value="TestPart" />
    <property name="responsePartName" value="TestPart" />
</action>
```

The ESB action class is *org.jboss.soa.esb.actions.bpel.BPELInvoke*.

The properties for this ESB action are:

- service

  This property is mandatory, and defines the service name registered in the WSDL associated with the deployed BPEL process.

- operation

  This property is mandatory, and represents the WSDL operation that is being invoked.

- requestPartName

This optional property can be used to define the WSDL message part that the inbound ESB message content should be mapped to. This property should be used where the ESB message does not already represent a multi-part message.

- responsePartName

  This optional property can be used to extract the content of a response multi-part WSDL message, and place this in the ESB message being passed to the next ESB action in the pipeline. If this property is not defined, then the complete multi-part message value will be placed in the ESB message.

This ESB action supports inbound messages with content defined as either:

- DOM

  If the message content is a DOM document or element, then this can either be used as the complete multi-part message, or as the content of a message part defined using the *requestPartName* property.

  If the message content is a DOM text node, then this can ONLY be used if a multi-part name has been defined in the *requestPartName* property.

- Java String

  If the message content is a string representation of an XML document, then the *requestPartName* is optional. If not specified, then the document must represent the multipart message.

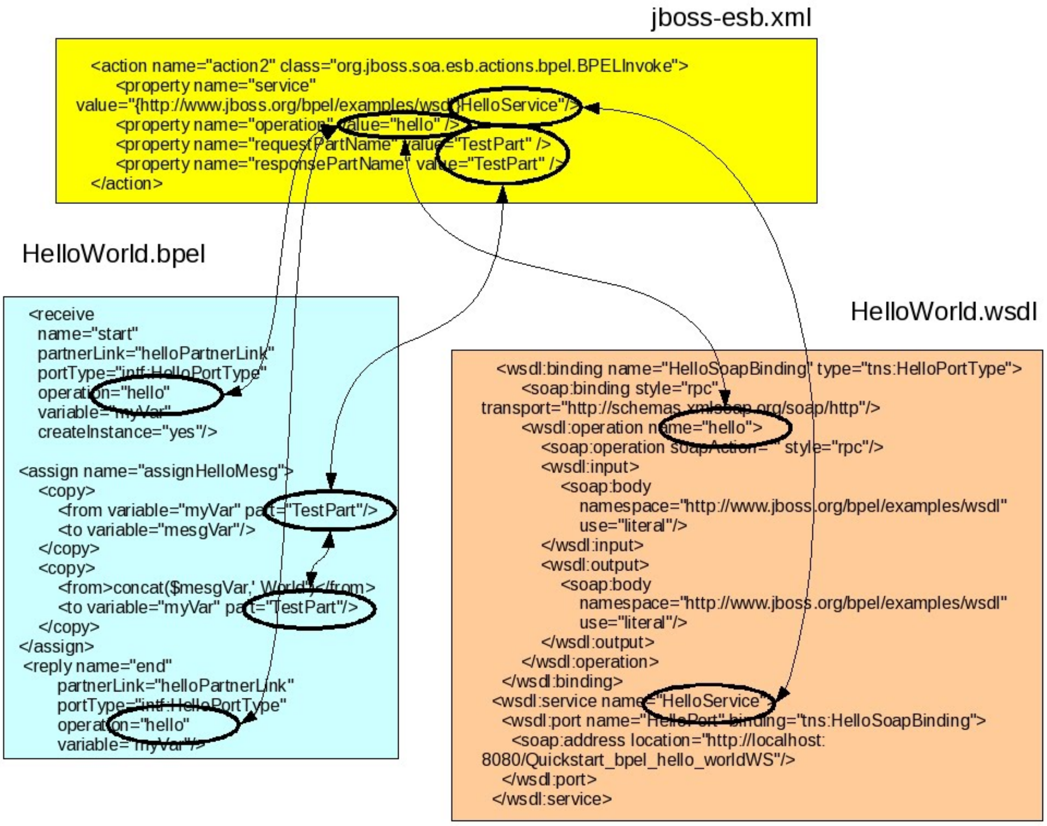
  If the message content is a string that does not represent an XML document, then the *requestPartName* must be specified.

When the message content represents the complete multipart message, this must be defined as a top level element (whose name is irrelevant) with immediate child elements that represent each of the multiple parts of the message. Each of these elements must then have a single element/node, that represents the value of the named part.

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

This shows an example of a multipart message structure. The top element (i.e. *message*) is unimportant. The elements at the next level represent the part names - in this case there is only a single part, with name *TestPart*. The value of this part is defined as a text node, with value "Hello World". However this could have been an element representing the root node of a more complex XML value.

The following diagram illustrates the inter-relationship of the JBossESB bpel_helloworld quickstart and the RiftSaw BPEL process configuration files.



```
<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="operation" value="hello" />
    <property name="requestPartName" value="TestPart" />
    <property name="responsePartName" value="TestPart" />
</action>
```

```
<receive
  name="start"
  partnerLink="helloPartnerLink"
  portType="intf:HelloPortType"
  operation="hello"
  variable="myVar"
  createInstance="yes"/>

<assign name="assignHelloMesg">
  <copy>
    <from variable="myVar" part="TestPart"/>
    <to variable="mesgVar"/>
  </copy>
  <copy>
    <from>concat($mesgVar,' World')</from>
    <to variable="myVar" part="TestPart"/>
  </copy>
</assign>
<reply name="end"
    partnerLink="helloPartnerLink"
    portType="intf:HelloPortType"
    operation="hello"
    variable="myVar"/>
```

```
<wsdl:binding name="HelloSoapBinding" type="tns:HelloPortType">
    <soap:binding style="rpc"
transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="hello">
      <soap:operation soapAction="" style="rpc"/>
      <wsdl:input>
        <soap:body
          namespace="http://www.jboss.org/bpel/examples/wsdl"
          use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body
          namespace="http://www.jboss.org/bpel/examples/wsdl"
          use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
 <wsdl:service name="HelloService">
  <wsdl:port name="HelloPort" binding="tns:HelloSoapBinding">
   <soap:address location="http://localhost:
8080/Quickstart_bpel_hello_worldWS"/>
  </wsdl:port>
 </wsdl:service>
```

4.2.1. Fault Handling

The normal response from a WSDL operation will be returned from the *BPELInvoke* ESB action as a normal message and placed on the action pipeline ready for processing by the next ESB action, or alternatively if no further actions have been defined, then returned back to the service client.

Faults, associated with a WSDL operation, are handled slightly differently. Depending on configuration it is possible to receive the fault as an ESB message or for the fault to be treated as an exception which aborts the action pipeline. The configuration property used to determine which behaviour is used is called *abortOnFault*. The default value for this property is "true". As an example, from the loan fault quickstart sample,

```
<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 has two relevant pieces of information, the fault type (or code) and the fault details. These are both returned in specific parts of ESB message's body.

1. Fault code (as 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, associated with the WSDL operation that was invoked.

   > **Warning**
   >
   > The specific version of the QName used by the JBoss server is from the stax-api.jar found in the server's lib/endorsed directory. If the client does not also include this jar in a folder that is in its endorsed directories, then a class version exception will occur when this ESB message part is accessed.

2. Fault code (as textual representation of QName)
   ESB message body part: *org.jboss.soa.bpel.message.fault.detail.code*

   This body part will return the textual representation of the QName for the fault code. The textual representation is of the form "{namespace}localpart" and can be converted back into a QName using 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 will contain the textual representation of the message content associated with the fault.