JBoss Portlet Bridge

Reference Guide

Release 1.0.0.B4

JBoss Portlet Bridge Overview

To get an idea of the JBoss Portlet Bridge community, the developers, and for wiki information, checkout *the project page* [http://www.jboss.org/portletbridge/].

What is the JBoss Portlet Bridge?

The JBoss Portlet Bridge (or JBPB for short) is an implementation of the *JSR-301* [http://jcp.org/en/jsr/detail?id=301] specification which supports JSF within a portlet and with added enhancements to support other web frameworks (such as *Seam* [http://www.seamframework.org/] and *RichFaces* [http://www.jboss.org/jbossrichfaces/]). It basically allows any Java developer to get started quickly with their JSF web application running in a portal environment. The developer no longer needs to worry about the underlying portlet development, portlet concepts, or the API.

Understanding how JSF works with Portal

The portlet bridge isn't a portlet. It's the mediator between the two environments and allows JSF and Portal to be completely unaware of each other and live in their own seperate worlds. The bridge is used to execute Faces requests on behalf of the portlet. During each request, the Faces environment is setup and handled by the bridge. Part of this implementation acts as a Faces controller much as the FacesServlet does in the direct client request world. The other part of this implementation is provided by implementating a variety of (standard) Faces extensions.

Getting started with JBoss Portlet Bridge

JBoss Portlet Bridge not only gives you the ability to run JSF web applications in a portlet, but also gives you the benefit of running supported JBoss frameworks like Seam and RichFaces.

1.1. Bridge Frameworks and Extensions

The JBoss Portlet Bridge currently supports JBoss Portal, JSF 1.2, JBoss Seam, and JBoss Richfaces. There are configurations that apply to supporting each framework. See section *Chapter 2, Bridge Configuration* for configuration instructions.

The JBoss Portlet Bridge project is also actively developing extensions, or Bridgelets, that enhance or bring together features of JBoss Portal, Seam, and Richfaces. For example, the PortalIdentity seam component allows you to drop the jar in your classpath and you instantly have SSO between Seam and Portal. This extension can also be configured with additional attributes in your Seam application's components.xml file.

Note

Don't forget that the bridge is still in Beta and so are any extensions. If you would like to contribute to any part of this project, we encourage you to be active on the *user forum* [http://www.jboss.com/index.html?module=bb&op=viewforum&f=273] and bring issues/enhancements to attention.

Table 1.1. Available Bridgelets

Bridgelet	Command

Single Sign On	By inlcuding the following dependency in your web pom, you will automatically have SSO between Jboss Portal and your seam application.
	
	`<dependency>`
	`  <groupId>org.jboss.portletbridge.extensions.seam</groupId>`
	`  <artifactId>PortalIdentity</artifactId>`
	`  <version>1.0.0-SNAPSHOT</version>`
	`</dependency>`

1.2. Before you start

Current version and compatibilty information can be easily located on the *JBPB wiki* [http://www.jboss.org/wiki/JBossPortletBridge]. Ensure you are using compatible versions of all integrated frameworks before you begin.

JBoss Portal provides it's latest distribution included in JBoss Application Server. All of the guesswork has been eliminated so that you can unzip and run Portal with a few clicks. *Get the latest here* [http://www.jboss.org/jbossportal/download/index.html] (ensure you choose the JBoss Portal + JBoss AS link)

Next, all that's left is to download the *JBoss Portlet Bridge distribution* [http://www.jboss.org/portletbridge/download/] and cofigure your portlet to use the bridge. Or, you can run a provided archetype *Section 1.3, "Maven Archetypes"* and deploy the generated war in a few easy steps. This will also give you an empty project to play around with or start from scratch.

For system requirements and setting up your JBoss Portal environment see the *reference guide* [http://docs.jboss.com/jbportal/v2.6.4/referenceGuide/html_single/#supportedversions].

1.3. Maven Archetypes

The JBPB project utilizes *Maven archetypes* [http://maven.apache.org/guides/introduction/introduction-to-archetypes.html] which allow you get up and running with different flavors of the bridge quickly.

Table 1.2. Available Archetypes

Archetype	Command
JSF 1.2 Basic	`mvn archetype:generate -DarchetypeGroupId=org.jboss.portletbridge.archetypes -DarchetypeArtifactId=1.2-basic -DarchetypeVersion=1.0.0.B4 -DgroupId=org.whatever.project -DartifactId=myprojectname -DarchetypeRepository=http://repository.jboss.org/maven2/`
RichFaces Basic	`mvn archetype:generate -DarchetypeGroupId=org.jboss.portletbridge.archetypes -DarchetypeArtifactId=richfaces-basic -DarchetypeVersion=1.0.0.B4 -DgroupId=org.whatever.project -DartifactId=myprojectname -DarchetypeRepository=http://repository.jboss.org/maven2/`

Archetype	Command
Seam Basic (Modular EAR)	`mvn archetype:generate -DarchetypeGroupId=org.jboss.portletbridge.archetypes -DarchetypeArtifactId=seam-basic -DarchetypeVersion=1.0.0.B4 -DgroupId=org.whatever.project -DartifactId=seamproject -DarchetypeRepository=http://repository.jboss.org/maven2/`

1.3.1. Running the Examples

JSF 1.2 Basic, RichFaces Basic, Seam Basic, and other demos

Each example application is configured to download the latest versions of JBoss Portal bundled with JBoss Application Server. After running the archetype *Section 1.3, "Maven Archetypes"* or *downloading the source code* [http://www.jboss.org/portletbridge/download/] for the example application that you're interested in, you can run one of the following Maven profiles to save time and get everything up and running with only 2 commands.

JBoss Portal 2.7.0.B1 + JBoss AS 4.2.2 (Bundled)

```
mvn install cargo:start -Premote-portal -Dportal-2.7.0.B1
mvn cargo:deploy -Premote-portal -Dportal-2.7.0.B1
```

JBoss Portlet Container 2.0 + JBoss AS 4.2.2 (Bundled)

```
mvn install cargo:start -Premote-portal -Dpc20
mvn cargo:deploy -Premote-portal -Dpc20
```

If you plan on using the cargo profiles to do active development, you can save alot of time by not downloading the bundle each time you do a clean install. To use a locally configured server bundled with portal, use the following command line parameters. The variable for JBOSS_HOME_DIR is related to how you zip *the server directory. If you zip the files under JBOSS_HOME/* then it will only be the name of your archive. But if you zip the actual folder JBOSS_HOME then JBOSS_HOME_DIR must be defined as 'zip file name/JBOSS_HOME folder name'.*

JBoss Portal 2.7.0.B1

```
mvn install cargo:start -Plocal-portal -DJBOSS_ZIP_HOME=/path_to_bundle_zip/
jboss-portal-2.7.0.B1-bundled.zip -DJBOSS_HOME_DIR=jboss-portal-2.7.0.B1-
bundled/jboss-portal-2.7.0.B1
mvn cargo:deploy -Plocal-portal -DJBOSS_ZIP_HOME=/path_to_bundle_zip/
jboss-portal-2.7.0.B1-bundled.zip -DJBOSS_HOME_DIR=jboss-portal-2.7.0.B1-
bundled/jboss-portal-2.7.0.B1
```

PortletContainer 2.0

```
mvn install cargo:start -Plocal-portal -DJBOSS_ZIP_HOME=/path_to_bundle_zip/
Jboss-4.2.2-PC20.zip -DJBOSS_HOME_DIR=Jboss-4.2.2-PC20
mvn cargo:deploy -Plocal-portal -DJBOSS_ZIP_HOME=/path_to_bundle_zip/
Jboss-4.2.2-PC20.zip -DJBOSS_HOME_DIR=Jboss-4.2.2-PC20
```

Bridge Configuration

The 301 specification is aimed at making the developers life as easy as possible with JSF+Portlet development. You will see below that there are minimal settings to getting any JSF web application up and running in the Portal environment.

2.1. Core Setup and Configuration

2.1.1. portlet.xml

The basic JSR-301 portlet configuration.

```
<portlet>
  <portlet-name>yourPortletName</portlet-name>
  <portlet-class>
    javax.portlet.faces.GenericFacesPortlet
  </portlet-class>

  <init-param>
    <name>javax.portlet.faces.defaultViewId.view</name>
    <value>/welcome.xhtml</value>
  </init-param>

  <init-param>
    <name>javax.portlet.faces.defaultViewId.edit</name>
    <value>/jsf/edit.xhtml</value>
  </init-param>

  <init-param>
    <name>javax.portlet.faces.defaultViewId.help</name>
    <value>/jsf/help.xhtml</value>
  </init-param>
```

When preserveActionParams is set to TRUE, the bridge must maintain any request parameters assigned during the portlet's action request. The request parameters are maintained in the*"bridge request scope"*. When this attribute isn't present or is FALSE the action's request parameters are only maintained for the duration of the*portlet request scope*.

```
<init-param>
  <name>javax.portlet.faces.preserveActionParams</name>
```

```
      <value>true</value>
    </init-param>
```

2.1.2. faces-config.xml

The PortletViewHandler ensures that each JSF portlet instance is porperly namespaced.

```
<faces-config>
   <application>
      <view-handler>
         org.jboss.portletbridge.application.PortletViewHandler
      </view-handler>
            <state-manager>org.jboss.portletbridge.application.PortletStateManager</state-
manager>
   </application>
   ...
```

2.1.3. Facelets Configuration

The following web.xml setting is only for Facelets based applications

2.1.3.1. web.xml

```
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/
web-app_2_4.xsd"
        version="2.4">
   ...
   <!-- This is optional parameters for a facelets based application -->
   <context-param>
      <param-name>org.ajax4jsf.VIEW_HANDLERS</param-name>
       <param-value>org.jboss.portletbridge.application.FaceletPortletViewHandler</param-
value>
   </context-param>
```

```
    <context-param>
```

```
        <param-name>javax.portlet.faces.renderPolicy</param-name>
        <param-value>
            ALWAYS_DELEGATE
        </param-value>
    </context-param>
    ...
</web-app>
```

RenderPolicy Options

- `ALWAYS_DELEGATE` Indicates the bridge should not render the view itself but rather always delegate the rendering.
- `NEVER_DELEGATE` Indicates the bridge should always render the view itself and never delegate.
- `DEFAULT` Directs the bridge to first delegate the render and if and only if an Exception is thrown then render the view based on its own logic. If the configuration parameter is not present or has an invalid value the bridge renders using default behavior. I.e. as if DEFAULT is set.

2.1.4. JSP Only Configuration

The following web.xml setting is only for JSP based applications

2.1.4.1. web.xml

```


<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/
web-app_2_4.xsd"
         version="2.4">

     <context-param>
        <param-name>javax.portlet.faces.renderPolicy</param-name>
        <param-value>
            NEVER_DELEGATE
        </param-value>
    </context-param>
    ...
```

```
</web-app>
```

2.1.5. JSR-301

The Jboss Portlet Bridge can be used with a any compatible implementation (for example, MyFaces implementation). Simply put the following into web.xml :

```


<context-param>
   <param-name>javax.portlet.faces.BridgeImplClass</param-name>
   <param-value>org.apache.myfaces.portlet.faces.bridge.BridgeImpl</param-value>
</context-param>
```

2.2. RichFaces Setup and Configuration Options

2.2.1. web.xml

The following configuration is designated for portlets using the RichFaces library. These settings will vary based on your individual needs. See *this section* [http://www.jboss.org/file-access/default/members/jbossrichfaces/freezone/docs/devguide/en/html/ArchitectureOverview.html#ScriptsandStylesLoadStrategy] of the RichFaces documentation for more details.

```

      <context-param>
         <param-name>org.richfaces.LoadStyleStrategy</param-name>
         <param-value>NONE</param-value>
      </context-param>
      <context-param>
         <param-name>org.richfaces.LoadScriptStrategy</param-name>
         <param-value>NONE</param-value>
      </context-param>

```

The `org.ajax4jsf.RESOURCE_URI_PREFIX` configuration cross references a setting in your `jboss-portlet.xml` file (see below). These settings are required for RichFaces.

```

      <context-param>
```

```
            <param-name>org.ajax4jsf.RESOURCE_URI_PREFIX</param-name>
            <param-value>rfRes</param-value>
        </context-param>

        <filter>
            <display-name>Ajax4jsf Filter</display-name>
            <filter-name>ajax4jsf</filter-name>
            <filter-class>org.ajax4jsf.Filter</filter-class>
        </filter>

        <filter-mapping>
            <filter-name>ajax4jsf</filter-name>
            <servlet-name>FacesServlet</servlet-name>
            <dispatcher>FORWARD</dispatcher>
            <dispatcher>REQUEST</dispatcher>
            <dispatcher>INCLUDE</dispatcher>
        </filter-mapping>
        ...
    </web-app>
```

2.2.2. jboss-portlet.xml

To avoid scripts loading more than once from different portlet windows you can define additional scripts in jboss-portlet.xml. *Note the "rfRes" URI prefix that is mapped in the web.xml.

```
<portlet>
    <portlet-name>ajaxPortlet</portlet-name>
    <header-content>
        <script src="/faces/rfRes/org/ajax4jsf/framework.pack.js" type="text/javascript"></script>
        <script src="/faces/rfRes/org/richfaces/ui.pack.js" type="text/javascript"></script>
        <link rel="stylesheet" type="text/css" href="/faces/rfRes/org/richfaces/skin.xcss"/>
    </header-content>
</portlet>
```

2.3. Seam Setup and Configuration Options

2.3.1. Configuration

The ExceptionHandler is used to clean Seam contexts and transactions after errors.

```
<context-param>
  <param-name>org.jboss.portletbridge.ExceptionHandler</param-name>
  <param-value>
    org.jboss.portletbridge.SeamExceptionHandlerImpl
  </param-value>
</context-param>
```

Earlier 2.0.x versions of Seam portlets must have the LIFECYCLE_ID set to SEAM_PORTLET in the web.xml. This setting allows the bridge to replace the original Seam phase listener during the faces lifecycle addPhaseListeners. This setting is not needed for Seam portlets version 2.1.x and up.

```
<context-param>
  <param-name>javax.faces.LIFECYCLE_ID</param-name>
  <param-value>SEAM_PORTLET</param-value>
</context-param>
```

Developing Portlets with the Bridge

This chapter demonstrates common development tasks described by the 301 specification.

3.1. Excluding Attributes from the Bridge Request Scope

When your application uses request attributes on a per request basis and you do not want that particular attribute to be managed in the extended bridge request scope, you must use the following configuration in your faces-config.xml. Below you will see that any attribute namespaced as foo.bar or any attribute beginning with foo.baz(wildcard) will be excluded from the bridge request scope and only be used per that application's request.

```
<application>
 <application-extension>
     <bridge:excluded-attributes>
         <bridge:excluded-attribute>foo.bar</bridge:excluded-attribute>
         <bridge:excluded-attribute>foo.baz.*</bridge:excluded-attribute>
     </bridge:excluded-attributes>
 </application-extension>
</application>
```

3.2. Supporting PortletMode Changes

A PortletMode represents a distinct render path within an application. There are three standard modes: view, edit, and help. The bridge's ExternalContext.encodeActionURL recognizes the query string parameter javax.portlet.faces.PortletMode and uses this parameter's value to set the portlet mode on the underlying portlet actionURL or response. Once processed it then removes this parameter from the query string. This means the following navigation rule causes one to render the \edit.jspx viewId in the portlet edit mode:

```
<navigation-rule>
   <from-view-id>/register.jspx</from-view-id>
   <navigation-case>
     <from-outcome>edit</from-outcome>
     <to-view-id>/edit.jspx?javax.portlet.faces.PortletMode=edit</to-view-id>
  </navigation-case>
</navigation-rule>
```

3.3. Navigating to a mode's last viewId

By default a mode change will start in the mode's default view without any (prior) existing state. One common portlet pattern when returning to the mode one left after entering another mode (e.g.. view -> edit -> view) is to return to the last view (and state) of this origin mode. The bridge will explicitly encode the necessary information so that when returning to a prior mode it can target the appropriate view and restore the appropriate state. The session attributes maintained by the bridge are intended to be used by developers to navigate back from a mode to the last location and state of a prior mode. As such a developer needs to describe a dynamic navigation: "from view X return to the last view of mode y". This is most easily expressed via an EL expression. E.g.

```
<navigation-rule>
  <from-view-id>/edit.jspx*</from-view-id>
  <navigation-case>
    <from-outcome>view</from-outcome>
    <to-view-id>#{sessionScope['javax.portlet.faces.viewIdHistory.view']}</to-view-id>
  </navigation-case>
</navigation-rule>
```

Note to Portlet Developers

Depending on the bridge implementation, when using values from these session scoped attributes or any viewIds which may contain query string parameters it may be necessary to use the wildcard syntax when identifying the rule target. For example, the above

```
<to-view-id>
```

expression returns a viewId of the form

```
/viewId?javax.portlet.faces.PortletMode=view&....
```

Without wildcarding, when a subsequent navigation occurs from this new view, the navigation rules wouldn't resolve because there wouldn't be an exact match. Likewise, the above edit.jspx

```
<from-view-id>
```

is wildcarded because there are navigation rules that target it that use a query string (

```
<to-view-id> /edit.jspx?javax.portlet.faces.PortletMode=edit </to-view-id>
```

). Developers are encouraged to use such wildcarding to ensure they execute properly in the broadest set of bridge implementations.

3.4. Custom Ajax Error Handling

By default, error handling is sent to a standard servlet page for Ajax requests. To handle the error inside the portlet, use the following javascript:

```
<script type="text/javascript">
  A4J.AJAX.onError = function(req,status,message){
    window.alert("Custom onError handler "+message);
  }

  A4J.AJAX.onExpired = function(loc,expiredMsg){
    if(window.confirm("Custom onExpired handler "+expiredMsg+" for a location: "+loc)){
      return loc;
    } else {
      return false;
    }
  }
</script>
```