JBoss Portlet Bridge Reference Guide

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 a final implementation of the JSR-329 [http://jcp.org/en/jsr/detail?id=329] specification which supports the JSF 1.2 runtime within a JSR 286 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 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. 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. What's New in 2.0?

1.1. Eventing

The bridge considers a portlet event a model event. I.e. the event is targeted to the applications data model not its view. As JSF events primarily concern its view, the bridge processes the portlet events manually, however provisions are made to make sure that any model changes that result from processing the event are updated in the view. Since event payloads are arbitrarily complex, the manual processing of the data, though managed by the bridge, is left to the (portlet) application to support.

See Section 4.1, "Sending and Receiving Events" for details and examples.

1.2. Portlet Served Resources

The bridge deals with portlet served resources in one of two ways. If the request is for a non-JSF resource, the bridge handles the request by acquiring a request dispatcher and forwarding the request to the named resource. If the request is for a JSF resource, the bridge runs the full JSF lifecycle ensuring that data is processed and the resource (markup) is rendered.

See Section 4.3, "Serving Your JSF Resources in a Portlet" for details and examples.

1.3. Public Render Parameters

The bridge automates the processing of public render parameters. A public render parameter can be mapped to an object's accessor (get/set method) designed to handle a String representation of the value via a Faces ValueExpression. When a new public render parameter value is received in a request, the bridge sets the value by calling the ValueExpression's setValue(). At the end of a request, if the current value of any mapped public render parameter doesn't match the current incoming value, the bridge sets the new value in an outgoing public render parameter (if feasible in the given phase).

See Section 4.2, "Public Render Parameters" for details and examples.

2. Bridge Frameworks and Extensions

The JBoss Portlet Bridge currently supports JBoss Portal, GateIn, JSF 1.2, JBoss Seam, and JBoss Richfaces. There are configurations that apply to supporting each framework. See section Chapter 3, Bridge Configuration for instructions.

The JBoss Portlet Bridge project is also actively developing extensions, and to differentiate from just another "project" that has boring ol' "extensions" we coined the term "Bridgelets" - because what would a project with Java and JSF be without having "*let" on the end of it? Not very cool in my opinion ;) With that said, in this current release we decided to bring all of our bridgelets into the impl code base since they are critical in most JSF portlet applications. Now it only takes a single line of configuration to utilize these features.

2.1. Seam Bridgelets

For example, the PortalIdentity seam component allows you to instantly have SSO between Seam and GateIn or JBoss Portal. This extension is configured in your Seam application's components.xml file as follows.

```
<security:portal-identity  authenticate-method="#{authenticator.authenticate}"/>
```

2.2. RichFaces Bridgelets

Richfaces does not account for multiple components on the same portal page by default. This following web.xml renders all RichFaces component javascript to be portal friendly.

```
<context-param>
   <param-name>org.jboss.portletbridge.WRAP_SCRIPTS</param-name>
   <param-value>true</param-value>
</context-param>
```

3. Before you start

Current version and compatibilty information can be easily located here [https://community.jboss.org/wiki/JBossPortletBridgeProjectLayout]. Ensure you are using compatible versions of all integrated frameworks before you begin.

GateIn provides it's latest distribution included in JBoss Application Server. All of the guesswork has been eliminated so that you can unzip and run the Portal with a few clicks. Get the latest here [http://jboss.org/gatein/downloads.html] (ensure you choose the 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 4, "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.

4. Maven Archetypes

Use the following archetype command to use any combination JSF 1.2, RichFaces, or Seam as a portlet.

```
mvn archetype:generate -DarchetypeCatalog=http://anonsvn.jboss.org/repos/portletbridge/tags/2.3.1.Final/archetypes/archetype-catalog.xml
```

5. Video Tutorials

Lesson 1: Getting Started With The Bridge [http://www.vimeo.com/3977469]

Lesson 2: Portlet 1.0 Advanced Seam and RichFaces [http://www.vimeo.com/4521877]

Lesson 3: Seam and Portlet 2.0 Eventing [http://www.vimeo.com/5847864]

Lesson 4: Running the 2.0 bridge on GateIn and deploy using JBoss Tools [http://www.vimeo.com/7255033]

Lesson 5: GateIn JMX Metrics and Dashboard Demo [http://www.vimeo.com/8752541]

Lesson 6: 2.0.0 FINAL and Best Practices for JSF IPC [http://www.vimeo.com/11484018]

Lesson 7: Running JSF Apps Over WSRP v2 [http://www.vimeo.com/16001694]

Check here for the latest videos. [http://www.vimeo.com/wesleyhales/videos]

Bridge Configuration

The 329 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.

If you are starting from scratch, we highly recommend you use the Section 4, “Maven Archetypes”.

1. Core Setup and Configuration

1.1. portlet.xml

The basic JSR-329 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>
```

1.2. faces-config.xml

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

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

1.3. Facelets Configuration

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

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.RENDER_POLICY</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.

1.4. JSP Only Configuration

The following web.xml setting is only for JSP based applications. Download the demo application here [http://anonsvn.jboss.org/repos/portletbridge/tags/2.3.1.Final/examples/jsf-ri/1.2-basic/].

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.RENDER_POLICY</param-name>
        <param-value>
          NEVER_DELEGATE
        </param-value>
      </context-param>
      ...
    </web-app>
```

1.5. JSR-329

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. RichFaces Setup and Configuration Options

2.1. web.xml

The following configuration is designated for portlets using the RichFaces library in GateIn. 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>ALL</param-value>
    </context-param>
    <context-param>
```

```
    <param-name>org.richfaces.LoadScriptStrategy</param-name>
    <param-value>ALL</param-value>
</context-param>
```

Note

If you use the "NONE" strategy, you must include the following scripts in your portlet or portal page header.

The **org.ajax4jsf.RESOURCE_URI_PREFIX** configuration cross references the path to your scripts below. These settings are required for RichFaces using the "NONE" strategy.

```
<a4j:loadScript src="resource:////org/ajax4jsf/framework.pack.js" type="text/javascript" />
<a4j:loadScript src="resource:////org/richfaces/ui.pack.js" type="text/javascript" />
<a4j:loadStyle src="resource:////org/richfaces/skin.xcss" />
```

Seam automatically configures your Ajax4JSF Filter, so if you are running a Seam portlet, you do not need the following Filter config. (But you do need the RESOURCE_URI_PREFIX no matter what)

```
<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>
```

3. Seam Setup and Configuration Options

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>
```

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

4. Portlet 2.0 Coordination

One very important thing to note before using either of the following mechanisms, is that you must have the proper 2.0 schema and xsd definition at the top of your portlet.xml.

```
<portlet-app xmlns="http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd"
      version="2.0"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd
  http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd">
```

4.1. Sending and Receiving Events

4.1.1. Configuration

Just like with any portlet 2.0 event consumer and receiver, you must define them in the portlet.xml. To see a working example, checkout the Seam Booking Demo portlet. http://anonsvn.jboss.org/repos/portletbridge/tags/2.3.1.Final/examples/seam/booking/

You must also define the following init params in your portlet.xml.

```
<init-param>
   <name>javax.portlet.faces.autoDispatchEvents</name>
   <value>true</value>
</init-param>
<init-param>
   <name>javax.portlet.faces.bridgeEventHandler</name>
   <value>org.foo.eventhandler</value>
</init-param>
```

For now, you must dipatch the event in the JSF or Seam backing bean. Future versions on the 2.0 bridge will automate the dispatching and consuming of events.

```
if (response instanceof StateAwareResponse) {
    StateAwareResponse stateResponse = (StateAwareResponse) response;
    stateResponse.setEvent(Foo.QNAME, new Bar());
}
```

Then you must also create the event handler class by implementing the BridgeEventHandler interface to process the event payload.

```
public class BookingEventHandler implements BridgeEventHandler
{
   public EventNavigationResult handleEvent(FacesContext context, Event event)
   {
       //process event payload here
   }
```

```
}
```

4.2. Public Render Parameters

4.2.1. Configuration

Public Render Parameters (or PRPs) are one of the most powerful and simple Portlet 2.0 features. Several portlets (JSF or not) can share the same render parameters. This feature can be use to present a cohesive UI to the user across all portlets on the page (i.e. using an employee ID to display relative data).

The bridge maps a render parameter to a backing bean using settings in your faces-config.xml and portlet.xml. A clear and working example can be found in the Seam Booking Demo portlet. http://anonsvn.jboss.org/repos/portletbridge/tags/2.3.1.Final/examples/seam/booking/

You must define the following init params in your portlet.xml.

```
<init-param>
  <name>javax.portlet.faces.bridgePublicRenderParameterHandler</name>
  <value>org.foo.PRPHandler</value>
</init-param>
...
<supported-public-render-parameter>myCoolPRP</supported-public-render-parameter>
```

Create a managed bean and public-parameter-mappings in your faces-config.xml. This should be a basic bean that you can bind the passed parameter to a string with getter and setter.

```
 <managed-bean>
    <managed-bean-name>bookingPRP</managed-bean-name>
    <managed-bean-class>your.class.Name</managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
</managed-bean>

<application>
   <application-extension>
      <bridge:public-parameter-mappings>
         <bridge:public-parameter-mapping>
            <parameter>"the name of your portlet":hotelName</parameter>
```

```
            <model-el>#{bookingPRP.hotelName}</model-el>
          </bridge:public-parameter-mapping>
        </bridge:public-parameter-mappings>
      </application-extension>
    </application>
```

You must set the parameter in the JSF or Seam backing bean, if you are providing one from your portlet.

```
  if (response instanceof StateAwareResponse) {
      StateAwareResponse stateResponse = (StateAwareResponse) response;
      stateResponse.setRenderParameter("hotelName",selectedHotel.getName());
   }
```

Then you must also implement the BridgePublicRenderParameterHandler interface to process any updates from the received parameter.

```
  public void processUpdates(FacesContext context)
  {
     ELContext elContext = context.getELContext();
       BookingPRPBean bean = (BookingPRPBean) elContext.getELResolver().getValue(elContext,
 null, "bookingPRP");

    if(null != bean){
      //Do something with bean.getHotelName());
    } else {

    }
  }
```

4.3. Serving Your JSF Resources in a Portlet

4.3.1. Configuration

We have setup a few examples to show you how to use EL and a simple bean that will allow you to use the portlet resource serving mechanism within a JSF portlet.

In ResourceBean.java [http://anonsvn.jboss.org/repos/portletbridge/tags/2.3.1.Final/examples/richfaces/richfaces-demo/src/main/java/org/richfaces/demo/common/ResourceBean.java], you can see a very simple implementations of a Map object that uses the bridge to get and encode a resource url served from the portlets web application.

So, when you have the normal "/images", "/styles" and other resource folders in your web app, you can use the following EL expression to serve them in your JSF application.

```
#{resource['/img/the-path-to-my-image.png']}
```

Just copy the ResourceBean.java code above, and add an entry to you faces-config.xml for the bean:

```
<managed-bean>
  <managed-bean-name>resource</managed-bean-name>
  <managed-bean-class>org.richfaces.demo.common.ResourceBean</managed-bean-class>
  <managed-bean-scope>application</managed-bean-scope>
 </managed-bean>
```

5. Additional Options

In the latest release we've added a couple of additional options to enable the portlet developer to better tailor the use of the JBoss Portlet Bridge to their needs.

5.1. WindowState Saving

As per the specification, by default the Bridge will retain the window state for subsequent requests.

This can potentially be problematic in those situations where you might have a Request Scoped Bean that has displayed a message. As the WindowState is retained, that message will remain until another user action on that Portlet is made, so we've now introduced a way to force the Bridge to disregard any existing WindowState data.

Adding the following to web.xml will allow your portlet to follow non Specification behavior:

```
    <context-param>
```

```
        <param-name>org.jboss.portletbridge.SAVE_WINDOWSTATE_AFTER_ACTION</param-name>
        <param-value>false</param-value>
    </context-param>
```

5.2. Restoring ViewId

Adding the following to web.xml will allow your portlet to be displayed with the initial page assigned to Mode.VIEW whenever a Render Phase is invoked, as opposed to the default which is the ViewId within the WindowState or Session:

```
    <context-param>
        <param-name>org.jboss.portletbridge.RESTORE_VIEWID</param-name>
        <param-value>false</param-value>
    </context-param>
```

Developing Portlets with the Bridge

This chapter demonstrates common development tasks described by the 329 specification.

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.

When using JSF client side state saving...

When javax.faces.STATE_SAVING_METHOD is set to "client", you must exclude the "com.sun.faces.*" attribute(s) in faces-config.xml. This can also be achieved by using the web.xml init-param "javax.portlet.faces.excludedRequestAttributes"

```
<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>
```

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. 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.

Using Seam Page Actions

Since Seam page actions are called during RENDER_RESPONSE, there is no guarantee how many times your portlet will be rendered. So page actions should only be used for navigation related tasks and never perform operations with side effects, ie. all page actions must be idempotent. For code that should only be called once in this scenario, component based actions should be used instead. They are called at the INVOKE_APPLICATION phase which is performed in the portal ActionRequest (one time per submit).

4. General Error Handling

Note

If you're developing a Seam portlet you can now use pages.xml for all error handling.

The following configuration may be used to handle exceptions. This is also useful for handling session timeout and ViewExpiredExceptions.

Pay attention to the location element. It must contain the /faces/ mapping to work properly.

```
<error-page>
  <exception-type>javax.servlet.ServletException</exception-type>
  <location>/faces/error.xhtml</location>
</error-page>
<error-page>
  <exception-type>javax.faces.application.ViewExpiredException</exception-type>
  <location>/faces/error.xhtml</location>
</error-page>
```

5. 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>
```

Also, add the following to web.xml. Read more about these settings here Request Errors and Session Expiration Handling [http://docs.jboss.org/richfaces/latest_3_3_X/en/devguide/html/ArchitectureOverview.html#RequestErrorsAndSessionExpirationHandling]

```
<context-param>
 <param-name>org.ajax4jsf.handleViewExpiredOnClient</param-name>
 <param-value>true</param-value>
</context-param>
```

6. Communication Between Your Portlets

There are roughly 4 different ways to send messages, events, and parameters between portlets which are contained in different ears/wars or contained in the same war. The Portlet Container does not care if you have 2 portlets in the same war or if they are separated, because each portlet has a different HttpSession.

Of course, with the Portlet 2.0 spec, the recommended way to share a parameter or event payload between 2 or more portlets are the Section 4.2, "Public Render Parameters" and Section 4.1, "Sending and Receiving Events" mechanisms. This allows you to decouple your application from surgically managing objects in the PortletSession.APPLICATION_SCOPE.

But, if these do not meet your usecase or you have a different strategy, you can use one of the following methods.

6.1. Storing Components in PortletSession.APPLICATION_SCOPE

Sometimes it makes sense to store your Seam components in the portlet APPLICATION_SCOPE. By default, these objects are stored in the PORTLET_SCOPE but with the annotation below, you can fish this class out of the PortletSession and use its values in other portlets across different Seam applications.

```
@PortletScope(PortletScope.ScopeType.APPLICATION_SCOPE)
```

Then you would pull the statefull object from the session:

```
                                                YourSessionClass      yourSessionClass      =
seamprojectPortletWindow?textHolder");
```

This method is demonstrated in this video: Lesson 2: Portlet 1.0 Advanced Seam and RichFaces [http://www.vimeo.com/4521877]

6.2. Using the PortletSession

If you need to access the PortletSession to simply share a parameter/value across multiple portlets, you can use the following to do so.

```
Object objSession = FacesContext.getCurrentInstance().getExternalContext().getSession(false);
try
{
  if (objSession instanceof PortletSession)
  {
    PortletSession portalSession = (PortletSession)objSession;
                                        portalSession.setAttribute("your    parameter    name","parameter
value",PortletSession.APPLICATION_SCOPE);
    ...
```

Then, in your JSP or Facelets page, you can use:

```
#{httpSessionScope['your parameter name']}
```

For more information about which EL variables are provided by the bridge, read section 6.5.1 of the JSR-329 specification [http://jcp.org/aboutJava/communityprocess/edr/jsr329/index2.html].

7. Linking to Portlet/JSF Pages Using h:outputink

For linking to any JSF/Facelets page within your portlet web application, you may use the following.

```
<h:outputLink value="#{facesContext.externalContext.requestContextPath}/home.xhtml">
  <f:param name="javax.portlet.faces.ViewLink" value="true"/>
  navigate to the test page
</h:outputLink>
```

8. Redirecting to an External Page or Resource

To link to a non JSF view (i.e. jboss.org) you can use the following parameter.

```
<h:commandLink actionListener="#{yourBean.yourListenr}">
  <f:param name="javax.portlet.faces.DirectLink" value="true"/>
  navigate to the test page
</h:commandLink>
```

Then in your backing bean, you must call a redirect().

```
FacesContext.getCurrentInstance().getExternalContext().redirect("http://www.jboss.org");
```

9. Using Provided EL Variables

All EL variables found in the JSR-329 (Portlet 2.0) specification are available in the JBoss Portlet Bridge. For example, you can use the following to edit the portlet preferences on the UI.

```
<h:form>
                                                    <h:inputText id="pref" required="true"
value="#{mutablePortletPreferencesValues['userName'].value}" />
    <h:commandButton actionListener="#{myBean.savePref}" value="Save Preferences" />
  </h:form>
```

Then in your backing bean, you must call the PortletPreferences.store() method.

```

```

```
Object request = FacesContext.getCurrentInstance().getExternalContext().getRequest();
PortletRequest portletRequest = (PortletRequest)request;
if (request instanceof PortletRequest) {
  try {
    PortletPreferences portletPreferences = portletRequest.getPreferences();
    portletPreferences.store();
```

10. Remote Portlet Navigation Using Portlet Events

When you send events using the method described here Section 4.1, "Sending and Receiving Events", you can also leverage the EventNavigationResult and return a JSF navigation rule. For Example, by returning a

```
new EventNavigationResult("fromAction","Outcome");
```

the fromAction can be a catch all like "/*" or a nav rule defined in your faces-config.xml and outcome can be the from-outcome node defined in the faces-config.xml navigation rule.