CONTENTS INCLUDE:

Mastering Portals with a Portlet Bridge

By Wesley Hales

ABOUT PORTLET BRIDGES

Portals have long been a thorn in the side of many developers who want to stay cutting edge—but must focus on the needs of the enterprise. And 99% of the time, a portal is the only answer for aggregating many applications into one UI.

A portlet bridge allows you to run application frameworks like JSF in a portal environment without worrying about the underlying portlet API or portlet concepts.

The JSR-301/329 portlet bridge boasts ease of transition, ease of initiation, and seamless mappings from JSF to the portal environment. Overall, a portlet bridge is a must-have mediator when working with portals, and it allows developers to learn a new technology painlessly.

ABOUT THE JBOSS PORTLET BRIDGE

This refcard shows specific configurations for the JBoss Portlet Bridge. The JBoss Portlet Bridge is a non-final draft implementation of the JSR-329 specification which supports the JSF 1.2 and 2.0 runtime within a JSR 168 or 286 portlet and with added enhancements to support other web frameworks (such as Seam and RichFaces).

The project follows the spec from JSR-301 to JSR-329 and now maintains an alpha version (3.0) compatible with JSF 2.0 and RichFaces 4.0.

Versions

Since this portlet bridge spans four different JSRs and supports two additional JSF frameworks, understanding versions can be a little daunting. i.e. Just because the portlet bridge is versioned at 2.0 does not mean it covers the JSF 2.0 specification.

JBoss Portlet Bridge Latest Versions	Compatible Frameworks
2.1.0.FINAL	JSR-286 Portlets, JSF 1.2, RichFaces 3.3.3.FINAL, Seam 2.2.1.CR2
3.0.0.ALPHA	SR-286 Portlets, JSF 2.0, RichFaces 4.0

IN THE BEGINNING, THERE WERE PORTLETS

It would only be fair (and cruel in some states) if I gave you a brief primer on portal and portlet technologies. There will come a day when you will need to understand what an ActionRequest is and what you can pull from your FacesContext.ExternalContext() while running as a JSF portlet.

Portal Terminology

Portal

Hosts the presentation layer for personalization, single sign on and content aggregation.

Portlet Container

A portal can host multiple portlet containers, and each portlet container has its own runtime environment.

Portal Page: Aggregation of portlets, organized and displayed on a single page.

Portlet: A portlet is a Java technology-based web component, managed by a portlet container, that processes requests and generates dynamic content.

Portlet Instance: A portlet instance can be placed on multiple pages and will show the same state in regards to the current mode.

Portlets vs. Servlets

Most rookie portlet developers have their roots in the servlet world. So it's important to know the key differences in what you're about to dive into from what you've been dealing with. Portlets generate a portion or "fragment" of the HTML page they're being displayed upon. This is the first big bullet point in understanding differences between servlets and portlets. UI wise, they cannot make use of the same html (for instance the standard <html> <head> <title> and <body> are forbidden in the portlet markup). The job of writing these tags belongs to the Portal Page and not you, the developer.

Points To Remember

- Portlet URLs, which are used as links to other pages within your Portlet window, are dynamically pre-generated by the Portlet code and passed into the markup.
- Although servlets and portlets are in some respects similar, their containers and lifecycles differ.
- On the front end, portlets are HTML fragments that compose a larger page which they are unaware of. Servlets provide the entire page.
- Portlets cannot be accessed by use of a single URL, the

way in which servlets are accessed. Instead, the URL points to the page where many portlets may be contained.

- The portal request paradigm is completely different. There are two requests instead of one. This is talked about in the next section.
- You now have mode and state buttons that allow you to "minimize" or "maximize" the portlet window or enter "help" mode.
- Portlets support two scopes within the session (PORTLET_SCOPE and APPLICATION_SCOPE). This allows for cases like sharing data from one portlet to another.
- Portlets are not allowed to set character encoding on the response, set HTTP headers, or manipulate the URL of the client request to the portal.

Portlet Lifecycle

Once you wrap your head around the fact that there are two requests and the Portal sends them for you, the rest is a piece of cake. What I'm about to explain makes total sense, but it's a huge change from the way things are done in a Servlet-based web app. In the Servlet world, a Servlet owns the process of receiving an entire web request and rendering output at the same time in one big response. In the Portlet world, the Portal receives the web request then makes separate calls to individual portlets. This separation allows for a different lifecycle method to be introduced in Portlets. The two requests are called **Action** and **Render,** and each have their own request and response. You must first keep in mind that when you click a link or submit button in the portlet window, it has been generated by the portal API. Thus, the portal needs to control/remember things like which window is sending the request, modes, parameters, window states, etc. So when an Action link is clicked, the portal will get this information and run processAction() along with a few other methods in your portlet. You are guaranteed that this action will be complete before the rendering phase starts. Once this is done, the portal now needs to re-render the page along with all the other portlets on that page. The best thing to keep in mind is portlets may be asked to render, even when the user is not interacting with them. Conceptually, this makes sense to have the two requests. However, developers have been naturally forced to do both of these things at the same time in the servlet world.

In most cases, when converting an existing servlet based application to a portlet, you must use a portlet bridge so that URLs are generated for the portlet world—along with other framework functionality.

GIVING PORTLETS A SECOND CHANCE

If You Skipped The First Page...

It's okay because the portlet bridge is created to mask all of those hairy details from the developer so you only have to worry about your web application. You can now take your existing JSF-based application and plug it into a portal painlessly.

Getting Started

Use the following maven archetype command:

```
mvn archetype:generate
-DarchetypeCatalog=http://bit.ly/jbossportletbridge
```

```
Choose archetype:
1: http://bit.ly/jbossportletbridge -> seam-basic ([2.1
2: http://bit.ly/jbossportletbridge -> richfaces-basic
3: http://bit.ly/jbossportletbridge -> 1.2-basic ([2.1.
Choose a number: :
```

Choose from the available projects and you will have everything you need to start developing a new portlet. If you are moving an existing application to a portlet, this will serve as a reference point for configuration.

Archetype Options

You will also see other configuration options like "make-remotable" and "richfaces-javascript-namespace" in the archetype generate phase.

Option	Purpose
make-remotable	Boolean property that will create the configuration for running this portlet remotely over WSRP
richfaces-javascript-namespace	Boolean property that allows for namespacing dynamically generated RichFaces javascript.

CONFIGURATION

The portlet bridge is a mediator between your web application and the portal worlds. It is not a portlet. So here we will review the changes you must make to convert your current JSF application into a portlet.

Core JSF Portlet Setup

portlet.xml

The following config is the backbone of your portlet. It shows where the supplied GenericFacesPortlet is and which JSF pages to render when clicking on the mode buttons (i.e. help, edit, etc...).

```
<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. You should set this as true if your application is not receiving request parameters that were set when a button or link was clicked.

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

faces-config.xml

The PortletViewHandler ensures that each JSF portlet instance is properly namespaced. Your application will use a combined portal and JSF namespace.

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

web.xml

The following setting tells your portlet to use the FaceletPortletViewHandler when using RichFaces.

```
<context-param>
    <param-name>org.ajax4jsf.VIEW_HANDLERS</param-name>
    <param-value>org.jboss.portletbridge.application.
FaceletPortletViewHandler</param-value>
</context-param>
```

ALWAYS_DELEGATE Indicates the bridge should not render the view itself but rather always delegate the rendering. This is the default setting for Facelets. If strictly using .jsp, then set the following to NEVER_DELEGATE.

```
<context-param>
    <param-name>javax.portlet.faces.RENDER_POLICY</param-name>
    <param-value>
        ALWAYS_DELEGATE
    </param-value>
</context-param>
```

RichFaces Portlet Configuration

web.xml

RichFaces maintains dynamic scripts and styles that are injected in the page at render. The recommended settings for a portal environment are:

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

RichFaces does not account for multiple components on the same portal page by default. This following parameter 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>
```

Seam automatically configures your Ajax4JSF/RichFaces Filter, so if you are running a Seam portlet, you do not need the following Filter config.

```
<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>
```

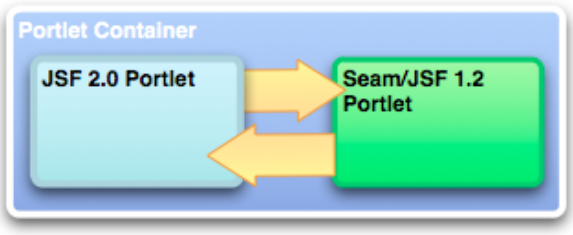
Seam Portlet Configuration

Hot Tip It's important to remember that your portlet may be rendered many times. Since Seam page actions are performed on the RenderRequest, you should only define your navigation in pages.xml.

The SeamExceptionHandler 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>
```

USING PORTLET 2.0 COORDINATION WITH JSF



Portlet 2.0 coordination gives us two ways to communicate between portlets (events and public render parameters). This means you can deploy a JSF 2 war and another JSF 1.2/Seam 2 ear and they can send events and messages to each other.

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">
```

Hot Tip Portlet events and public render parameters cannot be set during Ajax requests.

Sending and Receiving Events

The bridge is designed by default to defer to normal portlet processing when sending events. You should set the following parameters allowing portlet events to be sent directly to the bridge (autoDispatchEvents) and which class should process received portlet events (bridgeEventHandler).

```
<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 dispatch the event in the JSF or Seam backing bean. Future versions of the bridge should 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
    }

 }
```

Public Render Parameters

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 used 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 below.

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>hotelName</supported-public-
render-parameter>
```

Create a managed bean and public-parameter-mapping in your faces-config.xml. This should be a simple 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>portletName:hotelName</parameter>
          <model-el>#{bookingPRP.hotelName}</model-el>
       </bridge:public-parameter-mapping>
    </bridge:public-parameter-mappings>
  </application-extension>
</application>
```

Hot Tip The <parameter> must contain the name of your portlet followed by the parameter name defined in <supported-public-render-parameter> from the portlet.xml.

You can set the public render parameter in the JSF or Seam backing bean as follows:

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

And finally, in the receiving portlet, 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 {

}
}
```

Hot Tip You can find a working example using all coordination features running on JSF 1.2, RichFaces and Seam here: http://bit.ly/seambooking

SERVING JSF RESOURCES, THE PORTLET WAY

The below example shows how to create a simple bean that uses the portlet resource serving mechanism within a JSF portlet.

By creating a simple class (and defining it in your faces-config.xml) that implements the java.util.Map interface, you can create a "ResourceBean" that will make life much easier on the UI.

```
public String get(Object key) {
   FacesContext facesContext = FacesContext.getCurrentInstance();
   String url = null;
   if(null == key){
      url = null;
   } else if(null != facesContext){
      url = facesContext.getApplication().getViewHandler().
getResourceURL(facesContext, key.toString());
      url = facesContext.getExternalContext().encodeResourceURL(url);
   } else {
      url = key.toString();
   }
    return url;
   }
}
```

From here, you can serve images and other resources based in your web application by using:

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

The source for this example can be found here: http://bit.ly/resourcebean

DEVELOPER TIPS & TRICKS

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>
```

Hot Tip "From The Spec..." To support Faces, the bridge is responsible for encoding portlet (action) responses in a manner that allows it (and Faces) to reestablish the request environment that existed at the end of the corresponding action phase during subsequent render requests that are identified as pertaining to that action (or event). The term used to describe the scope of this managed data is the bridge request scope.

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>
```

Hot Tip PortletMode changes require an ActionRequest, so they cannot change modes during an Ajax or ResourceRequest.

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>
```

Hot Tip You should always wildcard your viewId's as shown in the above from-view-id. Developers are encouraged to use such wildcarding to ensure they execute properly in the broadest set of bridge implementations.

Clearing The View History When Changing Portlet Modes

By default, the bridge remembers the view history when you switch to a different portlet mode (like "Help" or "Edit"). You can use the following parameter in your portlet.xml to use the default viewId each time you switch modes.

```
<init-param>
   <name>javax.portlet.faces.extension.resetModeViewId</name>
   <value>true</value>
</init-param>
```

General Error Handling

Hot Tip If you're developing a Seam portlet, you can continue to 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>
```

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.

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

Hot Tip Remember to provide a link to the normal flow of your JSF application from your error page. Otherwise, the same error page will be displayed forever.

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 3.4.2, "Public Render Parameters" and Section 3.4.1, "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 use case or you have a different strategy, you can use one of the following methods.

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 = (YourSessionClass)
getRenderRequest().getAttribute("javax.portlet.p./default/
seamproject/seamprojectPortletWindow?textHolder");
```

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']}
```

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>
```

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");
```

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="#{mutablePortletPrefe
rencesValues['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();
```

Remote Portlet Navigation Using Portlet Events

When you send events, you can also leverage the EventNavigationResult and return a JSF navigation rule. For example, by returning:

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

The fromAction can be a wildcard "/*" 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.

CONCLUSION

With all the bitter sweetness that JSF and portal technologies offer, the portlet bridge does a good job of keeping everything in check. It also gives you possibilities that may not have been thought of when your JSF app was first created. Some examples are integration of legacy applications to leverage existing investments and the ability to develop newer frameworks and technologies without regression.

ABOUT THE AUTHOR

Twitter.com/wesleyhales

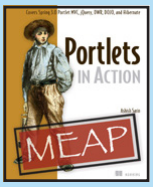
www.wesleyhales.com

Wesley Hales is a software developer on several projects at JBoss, by Red Hat. He is the co-founder of the JBoss Portlet Bridge project, a senior developer on GateIn, and serves as the JBoss representative on the JSR 301 & 329 specifications.

Before working for Red Hat, Wesley focused mainly on web framework and UI-related technologies, thus leading to committer roles for Apache, Mozilla, and JBoss. Wesley produces a series of screencasts on vimeo for each bimonthly release of the bridge. He has written a host of articles on infoq.com and posts occasional tutorials to his blog on jroller.com.

For his sins, he evangelizes portlet and portlet bridge technologies at most major conferences including Jazoon, AjaxWorld, JUDCon, DevNexus and whichever ones a daring enough to have a portal talk in their schedule.

RECOMMENDED BOOK



Portlets in Action is a comprehensive guide for Java developers with minimal or no experience working with portlets. Fully exploring the Portlet 2.0 API and using widely adopted frameworks like Spring 3.0 Portlet MVC, Hibernate, and DWR, it teaches you portal and portlet development by walking you through a Book Catalog portlet and Book Portal examples. The example Book Catalog Portlet, developed incrementally in each chapter of the book, incorporates most key portlet features, and the accompanying source code can be easily adapted and reused by readers. The example Book Portal application introduces you to the challenges faced in developing web portals.

BUY NOW

http://www.manning.com/sarin/

DZone communities deliver over 6 million pages each month to more than 3.3 million software developers, architects and decision makers. DZone offers something for everyone, including news, tutorials, cheat sheets, blogs, feature articles, source code and more. **"DZone is a developer's dream,"** says PC Magazine.

DZone, Inc.
140 Preston Executive Dr.
Suite 100
Cary, NC 27513

888.678.0399
919.678.0300

Refcardz Feedback Welcome
refcardz@dzone.com

Sponsorship Opportunities
sales@dzone.com

ISBN-13: 978-1-936502-32-5
ISBN-10: 1-936502-32-1
50795
9 781936 502325
$7.95

Version 1.0