Developing JSF Portlets with Rational Application Developer 6.0 and WebSphere Portal Server 5.1: Part 2, How to enable communication between portlets

Mikhail Genkin

January 18, 2007

This is the second part of a two-part series focusing on JSF (JavaServer Faces) and JSR (Java Specification Request) 168 development with IBM® Rational® Application Developer 6.0. Part 1 of the series looked at basic product features for JSF and JSR 168 development and created two portlets and three views for a CRMBrowser application. In this second part, you will learn how to enable communication between JSR 168 portlets to implement sophisticated screen flows.

View more content in this series

Introduction

This article is Part 2 in a two-article series (see the Resources section for Part 1) regarding portal development using IBM® Rational® Application Developer 6.0 (IRAD). As part of understanding this functionality, you will also learn about the basic JavaServer™ Faces (JSF) and IBM® WebSphere® Portal Server 5.1 features included with IRAD. Finally, this article will discuss how to:

- Expose portlet properties to the WebSphere Portal Server property broker via WSDL (Web Services Description Language)
- Pass complex, user-defined data types between portlets
- Override behavior of the portlet class generated by IRAD to switch your portlet view based on data sent by another portlet.

Developing cooperative portlets with the mix of technologies that you are using involves the following high-level steps:

1. Design how the portlets will interact - that is, describe how the two portlets will exchange data.
2. Enable the source portlet.
3. Enable the target portlet.
4. Enable different interactions that can take place between the source and target portlets.
5. Define wires between portlets.
6. Invoke actions that trigger property transfer via WebSphere Portal Server property broker.
First, we will start where we left off in Part 1, and complete these steps to implement full Summary-to-Detail browser functionality for our CRMBrowser application. You can download the sample code from the Download section in this article.

Desiging Portlet Interaction

Before describing how to enable the interaction between the Summary and the Detail portlets, this article will discuss how these portlets will interact at run-time. Whenever you click an action link in the Summary portlet, it will send a message to the Detail portlet telling it to display the corresponding view and data.

The Summary portlet will send different data to the Detail portlet depending on the action link you clicked. For example, if you click the customer id link, then the Summary portlet will send an instance of the com.ibm.sample.crmbrowser.CustomerDetailInfo bean to the Detail portlet. Alternatively, if you click the issue id link, then the Summary portlet will send an instance of the com.ibm.sample.crmbrowser.Issue bean to the Detail portlet.

In order to tell the WebSphere Portal Server property broker how the two portlets will interact, you need to define a wire between them. The wire is described using WSDL (this is discussed in more detail following). The source and target data type for the wire have to be the same; therefore, in order to pass your beans from the Summary portlet to the Detail portlet, you need to define an additional bean that will act as a holder for instances of the CustomerDetailInfo and Issue beans. An instance of the com.ibm.sample.crmbrowser.ShowDetailMessage bean will be transferred between the Summary portlet and the Detail portlet. It will hold exactly one instance of either the CustomerDetailInfo bean or the Issue bean, depending on which link you clicked in the CustomerSummaryView.jsp.

The following sections will describe how to configure the wire representing these semantics, as well as how to implement the required java code in the portlet, the source, and the target JSPs (JavaServer™ Pages).

Enabling the Summary portlet as a source

Enabling the source portlet involves the following steps:

1. Describe the source properties and actions with WSDL.
2. Enable the source action.
   - Configure parameters for the action control that will trigger the property transfer between the source and target portlets.
   - Write some action handler code to put the right information into the right portal programming model object (for example, the request attribute).

JSR 168 portlets can cooperate with each other by exchanging properties via the property broker. A WSDL file describes those properties that the source portlet can publish (or send) to the property broker. In the following process, you’ll learn about the tools that IRAD provides to support cooperative portlet development.
Describing the source with WSDL

To enable our Summary portlet as a property source, simply right-click the portlet in the Project Explorer view to display the context menu. Click **Cooperative > Enable Source**, as shown in Figure 1.

**Figure 1. Launching the Enable Cooperative Source wizard**

Once the Enable Cooperative Source wizard launches (see Figure 2), enter the illustrated values for the following:

- A name for the **Data type**
- The **Namespace** for the new data type
- What you want your parameter **Bound to**
The term parameter refers to how the value will be transferred from the source portlet to the target portlet. The choices are:

1. **None**: This setting implies that you will not specify the way the value will be passed, so the default behaviour for property broker will be used.

2. **Render Parameter**: Supports only strings. The string value will be bound to the RenderRequest object. The render parameter can be set during the action phase and retrieved during the render phase of the portlet lifecycle. It cannot be retrieved during the action phase.

3. **Request Parameter**: Supports only strings. The string value will be bound to the ActionRequest object, and can be retrieved during the action processing stage of the portlet lifecycle. The parameter value will be meaningless at the conclusion of request processing (that is, it will not persist beyond a single invocation).

4. **Request Attribute**: Supports any JavaBean™ type. The bean will be bound to the ActionRequest object. Lifecycle considerations are the same as for the Request Parameter.

5. **Session**: Supports any JavaBean type. The bean will be bound to the session object and will persist for the duration of the HTTP session between the client and the portal server.

You will bind your property value as an attribute of the request. This allows you to use a complex Java type to pass information between portlets, without degrading server performance by storing unnecessary information in the session.

The Enable Cooperative Source wizard generates a WSDL file (see Figure 3) that describes the portlet to the property broker. Note that the Summary portlet is tagged with a distinctive icon to indicate that it is a property source. The WSDL file contains the following sections:

- **Types**: This section describes data types (using XML schema) that can be emitted by the source portlet.
- **Messages**: This section describes messages that can be produced or consumed by the portlet.
- **Port Type**: This section describes the abstract interface of the portlet as seen by the property broker.
- **Binding**: This section describes how the abstract interface (port type) is implemented.
Enabling the source action

The Enable Cooperative Source wizard will automatically generate a simple type to represent the message (see Listing 1). You need to manually modify the generated WSDL to describe a complex type (this is because our portlets need to pass values -- rather than simple strings -- to each other). Listing 2 shows the definition for your complex type.

Listing 1. Generated type -- this type describes a simple string value

```xml
<xsd:schema targetNamespace="http://crmbrowser.sample.ibm.com"> 
  <xsd:simpleType name="ShowDetailMessageType">  
    <xsd:restriction base="xsd:string"></xsd:restriction>  
  </xsd:simpleType> </xsd:schema>
```

You need to change this simple type to accurately describe the message that the Summary portlet will send to the DetailPortlet. The WebSphere Portal Server Wiring tool will later use this type definition to define a wire between the source and the target.

Listing 2. Manually changed complex type

```xml
<xsd:complexType name="ShowDetailMessageType"> 
  <xsd:all>  
    <xsd:element name="customerDetail" type="tns:CustomerDetailType" minOccurs="0" maxOccurs="1"/>  
    <xsd:element name="issueDetail" type="tns:IssueType" minOccurs="0" maxOccurs="1"/>  
  </xsd:all> 
</xsd:complexType> 
</xsd:complexType> 
```

Developing JSF Portlets with Rational Application Developer 6.0 and WebSphere Portal Server 5.1: Part 2, How to enable communication between portlets
The complex types in your WSDL are metadata used by the property broker to define a wire. They should accurately describe the contents of the beans that you are passing from the Summary portlet to the Detail portlet (for the purpose of accurately describing wires and the function they perform). However, the property broker in WebSphere Portal Server 5.1 does <em>not</em> perform runtime checks to ensure that the beans that you pass map exactly to these abstract definitions.

In addition to changing the type definition, you need to make some changes to the generated binding information. Listing 3 shows the default binding generated by the tool, while Listing 4 shows manual changes that you need to make to enable cooperative behaviour.

**Listing 3. Generated binding information**

```xml
<binding name="Summaryportlet_Binding" type="tns:Summaryportlet_Service">
<portlet:binding />
<operation name="Summaryportlet">
<portlet:action type="standard">
<output>
<portlet:param name="CustomerDetailType" partname="CustomerDetailType_Output" boundTo="request-attribute" />
</output>
</operation>
</binding>
```

**Listing 4. Manual modifications to the binding information (note that manual changes have been highlighted in bold)**

```xml
<binding name="Summaryportlet_Binding" type="tns:Summaryportlet_Service">
<portlet:binding />
<operation name="Summaryportlet">
<portlet:action type="sendShowDetailResponse" name="sendShowDetailResponse" caption="Send.Detail.Response" description="Send Detail Response" />
<output>
<portlet:param name="detailInfo" partname="ShowDetailType_Output" boundTo="request-attribute" caption="Detail.Info.Type" description="Detail Info Type" />
</output>
</operation>
</binding>
```

Key changes to the generated WSDL description include the following:

- Adding the <strong>name</strong> attribute to the operation element
- Adding the <strong>class</strong> attribute to the <strong>param</strong> element of the binding (the class attribute specifies the Java type that will pass between portlets)
- Adding the optional <strong>caption</strong> and <strong>description</strong> attributes (this is necessary in order to use the wiring tool provided by WebSphere Portal Server 5.1)
Changes highlighted in Listing 4 are not simply stylistic, but are rather essential for making portlet messaging work. In addition to changes highlighted in the types in binding section in Listing 3 and Listing 4, it is recommended that you edit the messages and port type section of the file as well in order to give messages and operations more descriptive names. This will help greatly when constructing more complex user interfaces involving multiple portlets and wires among them.

If you choose to make these changes, you will need to ensure that the WSDL is valid by matching message and operation names in messages, port type, and binding sections -- as demonstrated in provided sample source (from the Download section in this article).

In order to finish enabling your source portlet, you need to complete two more steps:

1. Describe a key action parameter that will trigger the transfer of data to the property broker
2. Write action implementation code to put data into the request object

To accomplish the first step, open the CustomerSummaryView.jsp in the Page Designer, and select the Customer Id command hyperlink. Edit the parameters for that action in the properties view as shown in Figure 4.

**Figure 4. Entering parameters for the action that will trigger property value transfer via the WPS property broker**
The key parameter that you need to define is com.ibm.portal.propertybroker.action, the value of which must be equal to the value specified for the action name attribute of the operation binding (sendShowDetailResponse in Listing 4).

The remaining step involves putting your complex type -- the com.ibm.sample.crmbrowser.beans.ShowDetailMessage bean -- into the request object. To do so, generate an action handler method for the Customer Info hyperlink by clicking the Quick Edit icon in the Properties view (see Figure 5).

**Figure 5. Generating the action handler code for CustomerId**

![Generating the action handler code for CustomerId](image)

Complete the action handler implementation as shown in Listing 5 following.

**Listing 5. Putting the CustomerDetailInfo bean into the request object**

```java
public String doLink1Action() {
    // Determine which action link has been clicked and dispatch the corresponding
    // object to the property broker ShowDetailMessage
    message = new ShowDetailMessage();
    message.setCustomerDetail(getCustomerSummary().getCustomerDetailInfo());
    getRequestScope().put("detailInfo", message);
    return ";"
}
```

With these steps completed, your Summary portlet is effectively enabled as a property source. However, if you go back to the original application scenario, the Summary portlet can send two different types of messages to the Detail portlet:
• One to send customer detail information when the user clicks the `customerId` link
• The other to send detailed issue information when the customer clicks the `issueld` link

So essentially what you need to do is generate action handler code for the issueld link:

1. Select link in the page designer.
2. Enter parameters:

   • 1 -- enter the same value as for the `com.ibm.portal.propertybroker.action` parameter
   • 2 -- configure a parameter to retrieve the value of the originating action link

1. Write action handler code to populate the `com.ibm.sample.crmbrowser.beans.ShowDetailMessage` bean with an appropriate instance of the `com.ibm.sample.crmbrowser.beans.Issue` bean, and then store the former as an attribute of the request.

To create a parameter that will hold the value of the selected issue for step 2:

1. Click the **Add Parameter** button in the Properties view (shown in Figure 5).
2. Enter the name `issueId` in the left column of the Parameters table.
3. Click in the right column of the parameters table -- a button should appear in that column.
4. Click the button to bring up the Select Page Data Object dialog.
5. Navigate the objects shown in this dialog to find the `issueId` field of the `Issue` object contained in the `issues` array list (as shown in Figure 6).

![Figure 6. Mapping the issueld parameter to a variable](image)

This parameter will pass the selected issueld to our action handler code. The action handler code (shown in Listing 6) uses this parameter to retrieve the appropriate `Issue` object.
Listing 6. Sending the Issue via the request object.

```java
public String doLink2Action() {
    // Type Java code that runs when the component is clicked
    ShowDetailMessage message = new ShowDetailMessage();
    String issueId = (String)getRequestParam().get("issueId");
    if (issueId != null) { for (int i = 0; i <
            getCustomerSummary().getIssues().size(); i++)
        { Issue issue = (Issue)getCustomerSummary().getIssues().get(i);
            if (issueId.equals(issue.getIssueId()))
                { message.setIssueDetail(issue);
                    getRequestScope().put("detailInfo", message); break; } } }
    return "";
}
```

With these steps completed, your Summary portlet is fully enabled to serve as a property source to support your application design.

**Enabling detail portlet as target**

To enable the Detail portlet as a target for property values, you need to perform the following steps:

1. Generate the WSDL that describes the portlet interface to the WPS property broker.
2. Override the provided portlet class, and implement the `processAction()` method. This will process action requests from the property broker, and re-direct to corresponding views.
3. Modify the generated pagecode to initialize the model bean when the page is loaded.

**Generating the WSDL**

You will use the Enable Cooperative Target wizard to generate the WSDL. To launch the wizard, simply select the portlet in the project explorer view, right-click it, and click **Cooperative > Enable Target** (this step is nearly identical to enabling cooperative source).

The Enable Cooperative Target wizard is shown in Figure 7. It contains a number of fields that you need to fill in:

- **Data type**: Use the exact same name that you used when you enabled the source portlet. After the WSDL has been generated, furthermore, you'll need to manually edit it to ensure that the type definition in the target WSDL is identical to that in the source WSDL.
- **Namespace**: Again, the namespace that you enter here should be the same one you used for the source.
- **Action**: Enter any name you wish. Although you could use the browse button to locate the action on the target page, this approach is not recommended for this situation because the target portlet will switch the view dynamically. It is convenient (but not mandatory) to prefix the name of the action with `receive`.
- **Parameter**: You can use any name here to identify the type instance. You should probably use a name similar to the type name to help keep track of different type instances.
- **Bound to**: Values in this field are the same as described in the earlier section regarding enabling source. Lifecycle considerations are the same as well. In this case you will use the
**Request Attribute** setting. This ensures that your message (property value) will be delivered to the target portlet as an attribute of the `ActionRequest` object.

- **Label and Description**: These fields are optional, but should be filled in, as they will help you create the wires when you use the wiring tool. You can use any values for these fields.

**Figure 7. The Enable Cooperative Target wizard**

The Enable Cooperative Target wizard generates a WSDL file (see Figure 8) that describes the properties the target portlet can receive via the property broker. In order to enable the target you need to edit the generated WSDL manually:

- Change the generated type definition to be identical to that declared to for the source portlet. Source type and target type have to be identical; otherwise, the wiring tool will not allow you to create the wire between the two portlets during unit-testing.
- Edit the message, port-type, and binding information to reflect the conventions you use for message and operation naming (this is optional insofar as getting messaging to work is concerned, but is highly recommended for complex interfaces with many messaging possibilities).
- Edit the binding information.
Listing 7 shows the generated WSDL binding. Listing 7 shows the manual changes that you need to make to the binding information in order to enable cooperative behavior.

Listing 7. WSDL generated by the Enable Cooperative Target wizard.

```xml
<binding name="Detailportlet_Binding" type="tns:Detailportlet_Service">
  <portlet:binding />
  <operation name="Detailportlet">
    <portlet:action name="receiveShowDetailRequest" type="standard" caption="Receive.Detail.Request" description="Receive Detail Request" />
    <input>
      <portlet:param name="detailInfo" partname="ShowDetailMessageType_Input" boundTo="request-attribute" />
    </input>
  </operation>
</binding>
```

Listing 8. Manually edited WSDL sections (changes are highlighted in bold).

```xml
<binding name="DetailPortlet_Binding" type="tns:DetailPortlet_Service">
  <portlet:binding />
  <operation name="receiveShowDetailRequest">
    <portlet:action name="receiveShowDetailRequest" type="standard" caption="Receive.Detail.Request" description="Receive Detail Request" />
    <input>
      <portlet:param name="detailInfo" class="com.ibm.sample.crmbrowser.beans.ShowDetailMessage" partname="ShowDetailType_Input" boundTo="request-attribute" caption="Detail.Info.Type" description="Detail Info Type" />
    </input>
  </operation>
</binding>
```

The really important changes to the binding information (those that are not just following the convention of your choice) involve adding a class attribute that describes the
fully qualified java type that will be passed to the target by the property broker. This type must be the same as the one used to enable the source portlet. In this case it is com.ibm.sample.crmbrowser.beans.ShowDetailMessage. You also need to add the caption and description attributes to the param element. This will help when you go to create the wire.

Overriding default portlet behavior

Now that the WSDL description of the target property has been completed, you need to override the supplied portlet class and implement the processAction() method. The IRAD JSR 168 Faces portlet project provides a class to facilitate JSF portlet development -- com.ibm.faces.webapp.FacesGenericPortlet. This class can be found in the jsf-portlet.jar file.

Implementation of this class contains some limitations that disallow:

- Dynamic view switching
- Handling user-defined data types

In order to overcome these limitations, you need to override the processAction() method in the target portlet. In addition, you’ll write some custom code to handle messages coming in from the source portlet.

To create the method override, simply follow these steps.

1. Create a new package under the JavaSource folder (I use com.ibm.sample.crmbrowser.portlet in the example code).
2. Create a new Java class named DetailPortlet
4. Once the Java editor opens, click inside the class definition and press Ctrl+Space to bring up code assist.
5. Select processAction() inside the code assist window.

Listing 9 shows your DetailPortlet.processAction() method. In the method implementation you need to perform some key actions:

1. Look for the com.ibm.portal.propertybroker.action parameter in the ActionRequest. This tells you that the action was triggered by the property broker (as opposed to the end user clicking an action control inside one of the current portlet views).
2. Retrieve the complex type from the ActionRequest.
3. Determine which view needs to be shown based on the contents of this type.
4. Store your complex type in the session.
5. Redirect the portlet to the appropriate view. This is accomplished by setting the com.ibm.faces.portlet.page.view session attribute to point to the URL of the desired view JSP. FacesGenericPortlet implementation will use this attribute to show the correct view when processing the render phase of the request-response sequence.
6. Call the base class implementation to process all other events (preserve default behaviour for view actions).


```java
public void processAction(ActionRequest request, ActionResponse response) throws PortletException {
    // Retreive target action name String
    String actionName = (String)request.getParameter("com.ibm.portal.propertybroker.action");
    if (actionName != null) {
        // Invoke custom processing only if property broker action String
        String actionURL = "";
        if (actionName.equals("receiveShowDetailRequest")) {
            ShowDetailMessage detailInfo = (ShowDetailMessage)request.getAttribute("detailInfo");
            if (detailInfo.getCustomerDetail() != null) actionURL = "/CustomerDetailView.jsp";
            else actionURL = "/IssueDetailView.jsp";
            request.getPortletSession().setAttribute("detailInfo", detailInfo);
        }
        if (!actionURL.equals("") && request.getPortletSession().getAttribute("com.ibm.faces.portlet.page.view")) {
            request.getPortletSession().setAttribute("com.ibm.faces.portlet.page.view", actionURL);
        }
    } else {
        super.processAction(request, response);
    }
}
```

Modifying the generated pagecode

The only remaining piece of coding is to make sure that the target view initializes properly when it is loaded by the portlet. Following the JSF architecture, the best place to do this would be when the bean that acts as the model for the JSF page is created. IRAD JSF tools generate JavaBeans that allow you to interact programmatically with JSF controls (as shown in Figure 9).

Figure 9. Package and beans generated by JSF tools

Listing 10 shows the implementation of the `IssueDetailView.getIssue()` method in `IssueDetailView.java`. This method will be called every time the JSF run-time reconstructs the objects tree that will eventually generate an HTML page for the end user. The key aspects to note in this method implementation are:
• Initialization is performed only if issue is null
• Our complex type -- the Issue -- is retrieved from the session

Listing 10. Implementing the pagecode.IssueDetailView.getIssue() method.

```java
public Issue getIssue() { if (issue == null) {
  ShowDetailMessage detailInfo = (ShowDetailMessage)getSessionScope().get("detailInfo");
  if (detailInfo != null) issue = detailInfo.getIssueDetail();
  if (issue == null) issue = new Issue(); } return issue; }
```

In order for the application to work fully, a similar method implementation needs to be written for pagecode.CustomerDetailView.getCustomerDetail(). (For further information, please see the included code from the Download section in this article).

Wiring the portlets and unit testing

The only step left is to unit test the application. If you have not already done so, create a new server by completing the following steps:

1. In the Servers view, right-click to bring up the context menu, then click New > Server.
2. Select WebSphere Portal 5.1 Unit Test Environment in the New Server wizard.
3. Add the CRMBrowserEAR to list of configured applications for this server.
4. Click Finish to exit the New Server wizard.
5. Double-click the new server in the Servers view to open the Server Configuration Editor.
6. Switch to the Portal tab, and select the Enable base portlets for portal administration and customization checkbox.
7. Save your changes and exit the editor.
8. Select the CRMBrowser dynamic Web project in the Project Explorer view.
9. Right-click to bring up the context menu, then click Run > Run On Server.

Your WebSphere Portal UTE will now start, and a browser showing your portlets will launch automatically.

Next you need to define a wire between your Summary portlet, and Detail portlet. To do this, complete the following steps:

1. Click the Edit Page tab of the portal interface shown in the browser.
2. Click the Wires tab.
3. Use drop down boxes in the Wires page to define a wire between the Summary portlet (source) and the Detail portlet (target). You do this by first selecting the Source Portlet, then the Sending action, then the Receiving Portlet, and finally the Receiving action in the appropriate drop-down boxes (as illustrated in Figure 10 following).
4. Click Done to exit this page and return to our application UI.
You are now ready to start using the CRMBrowser application. Views in the Detail portlet should change -- displaying either customer detail information or issue detail information -- depending on which action link you click in the Summary portlet view.

Conclusions

The WebSphere Portal Server Property Broker allows you to pass complex data between portlets. You can use this capability to implement sophisticated user interface flows between portlets. IRAD provides tooling that helps you leverage Property Broker capabilities during development. You can override code generated by IRAD to enhance its capabilities and enrich application behavior. To learn more about these exciting technologies, please browse other articles and references listed in the Resources section below.
Downloadable resources

<table>
<thead>
<tr>
<th>Description</th>
<th>Name</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code supporting article</td>
<td>crmbrowser.zip</td>
<td>2.61 MB</td>
</tr>
</tbody>
</table>

© Copyright IBM Corporation 2007

Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)