BEA WebLogic Portal™

Establishing Interportlet Communications

Version 8.1 Service Pack 5
Revised: October 2005

Contents

4. Establishing IPC by Using Custom and Page Flow Events

Overview of Interportlet Communications

Interportlet communications (IPC)—also called portlet-to-portlet communications—refers to how an event in one portlet controls some aspect of behavior in another portlet. Some examples of interportlet communications are:

- When maximizing one portlet causes another portlet to change its displayed mode from View to Edit.
- Selecting an item in a menu portlet will launch a page flow action in a visible content portlet, causing that portlet's contents to change.

Service Pack 4 of BEA WebLogic Portal 8.1 introduced a relatively simple tool for implementing interportlet communications. This tool—a dialog box in WebLogic Workshop—allows you to easily add event handlers to a portlet and then add actions for those handlers to fire when they detect the event on another portlet.

This document described how to establish IPC by implementing event handlers. It contains these subjects:

- The WebLogic Portal IPC Model
- How IPC is Implemented

The WebLogic Portal IPC Model

Earlier versions of WebLogic Portal allowed you to establish interportlet communications by using such techniques are adding `listenTo()` methods or backing files on page flow portlets. WebLogic Portal 8.1 with Service Pack 4 introduced a new IPC model based upon event handlers, Java objects that listen for predefined events on other portlets in the portal and fire actions when that event occurs.

Event Handlers

Event handlers "listen" for events raised on subscribed portlets and fire an action when a specific event is detected. Event handlers can listen and react to the following types of events:

- Generic Events
- Portal Events
- Custom Events
- Page Flow Events
- Struts Events

Portal Event Actions

Event actions depend upon the type of event being raised. Except for portal events, all other events can be identified in the Events field on the Event Handler tool. Events available with the portal event handler are listed in Table 1-1.

Table 1-1 Events Available to a Portal Event Handler

This event...	Fires an action when the portlet...
onActivation	Becomes visible.
onDeactivation	Ceases to be visible.
onMinimize	Is minimized
onMaximize	Is maximized
onNormal	Returns to its normal state from either a maximized or minimized state
onDelete	Is deleted from the portal.
onHelp	Enters the help mode
onEdit	Enters the edit mode

Table 1-1 Events Available to a Portal Event Handler

onView	Enters the view mode
onRefresh	Is refreshed

Custom Events

A custom event is an event that you define; in other words, it is not an event provided out-of-the-box with WebLogic Portal 8.1. A custom event can pass a developer-defined payload or fire any other predefined action. Custom events can be fired declaratively or based on a methods called in a backing file. A user will be able to specify that an event should be handled by a method in a backing file.

Event Actions

The event handlers fire an action on the listening portlets when that handler detects an event from another portlet in the application; for example, when the user minimizes the appropriate portlet a portal event called onMinimize might cause the handler listening for it to fire an action that invokes a backing file attached to it.

Table 1-2 lists the event actions available.

Table 1-2 Event Actions

This action...	Does this...
Change Window Mode	Changes the mode from its current mode to a user-specified mode; for example, from help mode to edit mode.
Change Window State	Changes the state from its current state to a user-specified state; for example, from maximized to delete state.
Activate Page	Opens a user-specified page.
Fire Generic Event	Fires a user-specified generic event.

Table 1-2 Event Actions

This action...	Does this...
Fire Custom Event	Fires a user-defined custom event. This event needs to be included in the portlet file.
Invoke BackingFile Method	Runs a method in the backing file attached to the portlet. Backing files allow you to programatically add functionality to a portlet to enable preprocessing (for example, authentication) prior to rendering the portal controls. For more information, please refer to Chapter 5, "Understanding Backing Files".

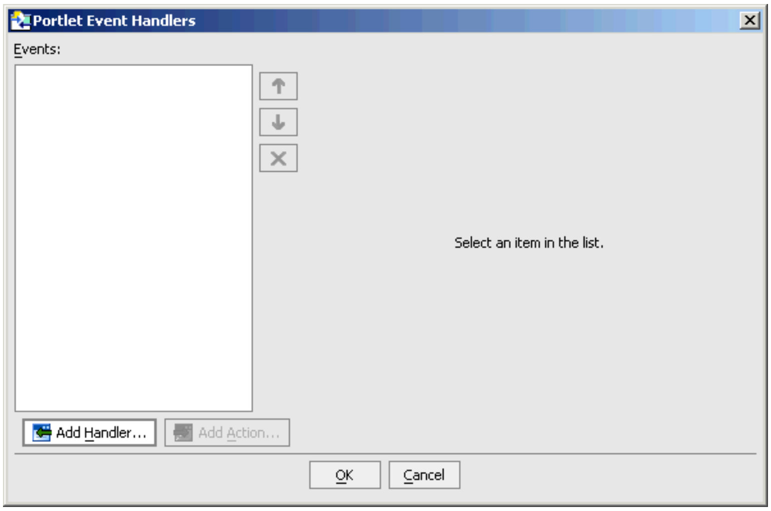
How IPC is Implemented

The IPC Tool included in WebLogic Workshop makes implementing event handlers relatively easy. To launch the tool, do the following:

1. Open a portlet in WebLogic Workshop.
2. In the Property Editor for that portlet, click the ellipses button (...) next to Event Handlers (if no event handlers have been added, the Event handler field will show that. If any event handlers have been added, the field will indicate the number added).

   The tool will appear, as shown in Figure 1-1.

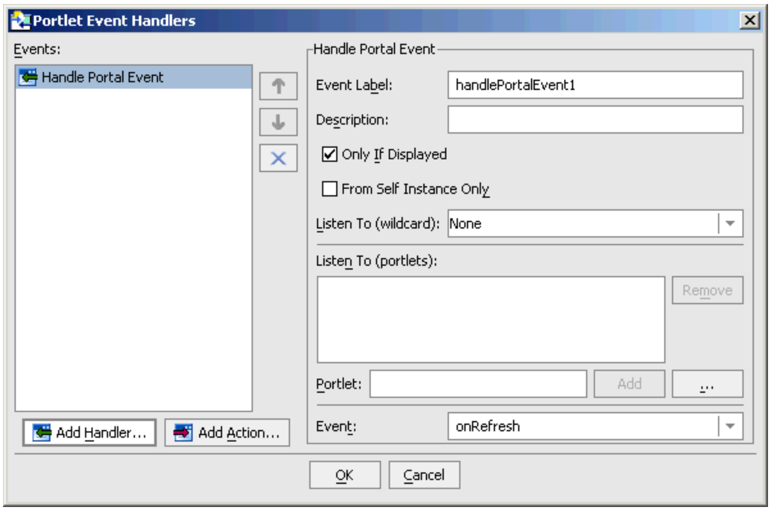
Figure 1-1 Event Handler Tool



3. Click Add Handler to open the event handler drop-down menu and select a handler.

   The dialog box will expand, opening up additional fields you can use to set up the handler (Figure 1-2).

Figure 1-2 Expanded Event Handler Tool



Event Label: handlePortalEvent1

Description:

[✓] Only If Displayed

[ ] From Self Instance Only

Listen To (wildcard): None

Listen To (portlets):

Portlet:

Event: onRefresh

The entire process of setting up an event handler can all be handled by using this tool. What you need to do is:

1. Select an event handler.
2. Determine the portlet(s) to which that handler will listen.
3. Select an event that the handler will listen for.
4. Select and configure an action to fire when the event occurs.
5. Save the event handler

For specific details on using the event handler tool, please refer to How Do I: Establish Interportlet Communications with WebLogic Workshop? in the WebLogic Workshop online help.

Setting Up the Examples

This manual contains two exercises that will step you through the procedures for establishing interportlet communications. One example uses a portal event handler (Chapter 3, “A Simple Example of Establishing IPC”), while the other employs both a custom event handler and a page flow event handler (Chapter 4, “Establishing IPC by Using Custom and Page Flow Events”). These examples will familiarize you with the workings of the Event Handler tool in the context of some common uses of that tool.

These examples are specific to interportlet communications within a single portal web application. They do not apply to federated portal applications. For information on establishing IPC with federated portals (such as WSRP), please refer to Establishing Interportlet Communications with Remote Portlets

This section includes information on the following topics:

- Parts of a Portal Application
- Step 1: Create the Domain
- Step 2: Create the Enterprise Application
- Step 3: Create the Portal Web Application (Project)
- Summary
- Next Steps

Parts of a Portal Application

A portal application is part of hierarchy of the components described in this section.

- Portal Domain: The portal domain is a logically related group of WebLogic Server resources that contain the application server, including the administration server and managed servers, used by the portal, represented by a `config.xml file`.
- Portal Application: This component, also called the "enterprise" application, consists of one or more portal web application (or "project") modules, EJB modules, and resource adapters. It might also include a client application. An enterprise application is defined by an `application.xml` file, which is the standard J2EE deployment descriptor for enterprise applications. You can have more that one portal application per domain.
- Portal Web Application: Alternately referred to as the "portal project," this component is the specific portal. It contains the elements of the portal framework necessary to build and render the portal, such as the WebLogic Portal JSP tags, APIs and the default framework files. The

In this section, you will create the domain and enterprise application that you will use in the exercises included in Chapter 3, "A Simple Example of Establishing IPC" and Chapter 4, "Establishing IPC by Using Custom and Page Flow Events". In this exercise, the root directory for your domain should be *`BEA_HOME`*`\user_projects\domains`; for the applications and all of their components, it's *`BEA_HOME`*`\user_projects\applications` (where *`BEA_HOME`* is the directory where you installed BEA WebLogic Platform`)`. These are default directories that BEA WebLogic Platform creates upon installation.

Step 1: Create the Domain

To create a domain, do the following:

1. Launch WebLogic Workshop.
2. Open the Tools menu and select **WebLogic Server**>**Configuration Wizard**.

   The Configuration Wizard launches.
3. Follow the prompts using the value specified in Table 2-1. Click Next when you are finished with each dialog box.

Table 2-1 Configuration Wizard Values

On...	Select or Enter...
Create or Extend a Configuration	Create a new WebLogic Configuration
Select a Configuration Template	Basic WebLogic Portal Domain
Choose Express or Custom Configuration	Express
Configure Administrative Username and Password	Username: `weblogic` Password: `weblogic` Confirm Password: `weblogic`
Configure Server Start Mode and Java SDK	JRockit SDK 1.4.2_05
Create WebLogic Configuration	Configuration Name: ipcDomain

4. Once the domain is created, click Done.

Step 2: Create the Enterprise Application

To create a portal application, do the following:

1. Open the File menu and select New>Application...

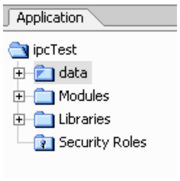
   The New Application dialog box appears.

2. Select Portal Application and do the following:
   - In Name, enter ipcTest.
   - In Server, click Browse to display the Select WebLogic Server config.xml File dialog box. Navigate to the ipcDomain directory and select `config.xml`.

3. Click Create.

   When the application is successfully created, it will appear in the application panel, as shown in Figure 2-1.

Figure 2-1 . Portal Application Created

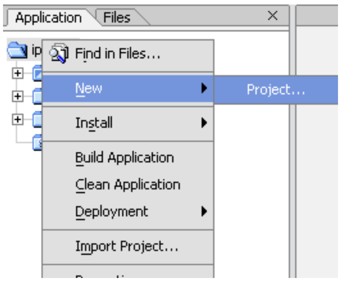


Step 3: Create the Portal Web Application (Project)

The portal web application, or portal project, will contain the framework components required to build and render the portal. To create the portal web application, use the following procedure:

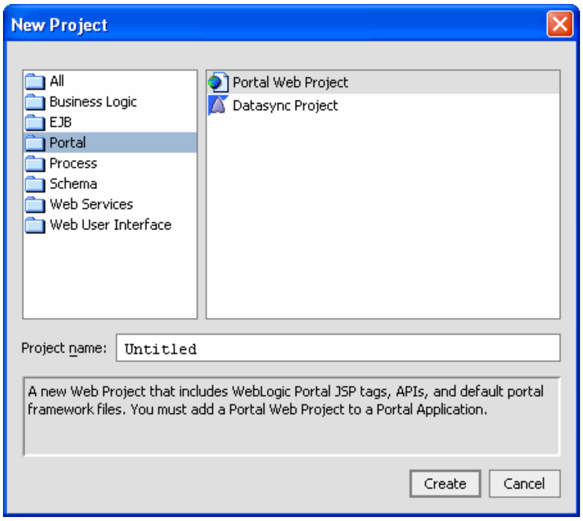
1. In the file tree, right-click the application name (ipcTest) and select **New**>**Project**, as shown in Figure 2-2.

Figure 2-2 Creating a New Project



The New Project dialog box appears (Figure 2-3):

Figure 2-3 New Project Dialog Box



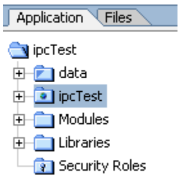
Project name: Untitled

A new Web Project that includes WebLogic Portal JSP tags, APIs, and default portal framework files. You must add a Portal Web Project to a Portal Application.

2. In the left pane of the dialog box, select Portal and, in **Filename**, enter `ipcTest`. Click **Create**.

   The project is updated (you will see a progress meter). Once the update is complete, the project appears as a directory in the file tree (Figure).

Figure 2-4 Project Added to File Tree



Summary

Upon completion of the preceding procedures, you will have created a domain, an enterprise application, and a portal web application. The file tree in the left-hand pane of WebLogic Workshop should look like the example in Figure 2-1. These will be the domain and enterprise application you will use for the exercises in this document.

Next Steps

With a development environment completed, you next will complete the exercises described in their respective chapters:

- Chapter 3, “A Simple Example of Establishing IPC”
- Chapter 4, “Establishing IPC by Using Custom and Page Flow Events”

In these exercises, you will create individual page flows, portlets, JSPs, and backing files to establish interportlet communications within the portal project. You will then add these portlets to a portal and test the application to ensure IPC has been achieved.

A Simple Example of Establishing IPC

This section describes the process of setting up interportlet communications between two portlets by using the Event handler tool in BEA WebLogic Workshop. This is a simple example in which minimizing one portlet will change the text string in another portlet in the portal.

You should become familiar with the Event Handler tool before attempting to replicate this example. Please refer to How Do I: Establish Interportlet Communications with WebLogic Workshop? in the BEA WebLogic Workshop online help system for more information.

This exercise is comprised of three main steps:

- Step 1: Create the Portlets
- Step 2: Test the Application

Before You Begin

If you have not set up your development environment as described in Chapter 2, "Setting Up the Examples," please refer to that section and follow these steps:

- Step 1: Create the Domain
- Step 2: Create the Enterprise Application
- Step 3: Create the Portal Web Application (Project)

After completing these steps, you will have portal domain called ipcDomain and a portal application called ipcTest.

Step 1: Create the Portlets

In this step, you will create two JSP files and the JSP portlets that will surface these files. You will also create a backing file that will contain the instructions necessary to complete the communication between the two portlets (for more information on backing files, please refer to Chapter 5, "Understanding Backing Files") and add an event handler to one of the portlets. Once you have created the portlets and attached the backing file, you will test the application in your browser.

Note: Before continuing with this procedure, ensure that WebLogic Workshop is running and the ipcTest web application node is expanded.

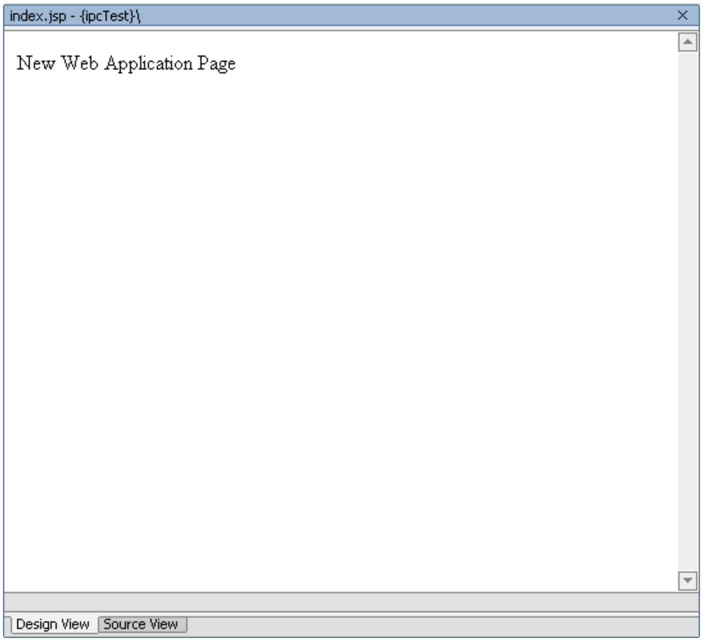
Create the JSP FIles and Portlets

To create the JSP files the portlets will surface, do the following:

1. Under the ipcTest node, double-click `index.jsp`.

   `index.jsp` opens in Design View (Figure 3-1).

Figure 3-1 index.jsp in Design View

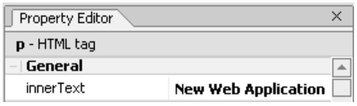


New Web Application Page

2. Click the phrase New Web Application Page to highlight it.

   A box will appear around the text and an inner text field will appear in the Property Editor, under General (Figure 3-2).

Figure 3-2 Properties Menu inner Text Field



Property Editor

p - HTML tag

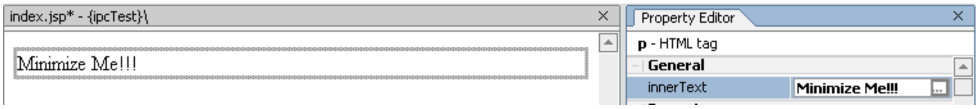
General

innerText	New Web Application

3. In the inner text field, click the ellipses button (...) to open the inner text dialog box and replace New Web Application Page with the phrase Minimize Me!!! Click OK.

   The dialog box closes and Minimize Me!!! appears in the inner text field and in the Design View, as shown in Figure 3-3.

Figure 3-3 New Text Added to the JSP File



index.jsp* - {ipcTest}\

Minimize Me!!!

Property Editor

p - HTML tag

General

innerText	Minimize Me!!!

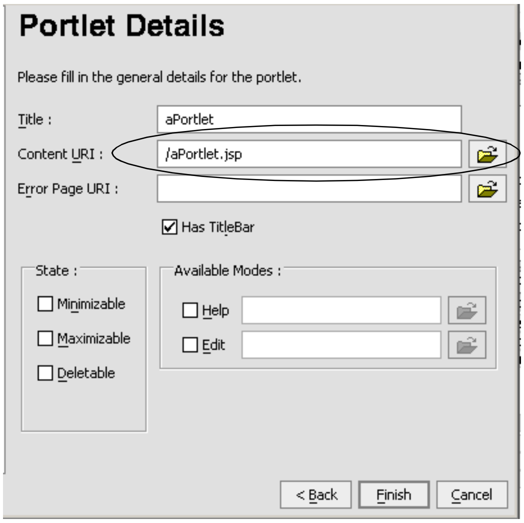
4. Open the File menu and select Save As...

5. Save the file as `aPortlet.jsp`.

6. Right click `aPortlet.jsp` in the Application tree and select Generate Portlet... from the context menu.

   The Portal Details dialog box appears (Figure 3-4). Note that `aPortlet.jsp` appears in the Content URI field.

Figure 3-4 Portal Details Dialog Box for a Portlet



7. Select Minimizable, Maximizable, and Deletable and click Finish.

   `aPortlet.portlet` will appear under ipcTest in the application tree.

8. `aPortlet.jsp` should still be open. If it isn't, reopen it.

9. Open the File menu and select Save as.

10. In the Name field of the Save "aPortlet.jsp" as dialog box, enter `bPortlet.jsp` and click Save.

11. On the JSP display, click Source View.

    The XML code for the JSP file appears.

12. Copy the code from Listing 3-1 into the JSP, replacing everything from `<netui:html>` through `</netui:html>`.

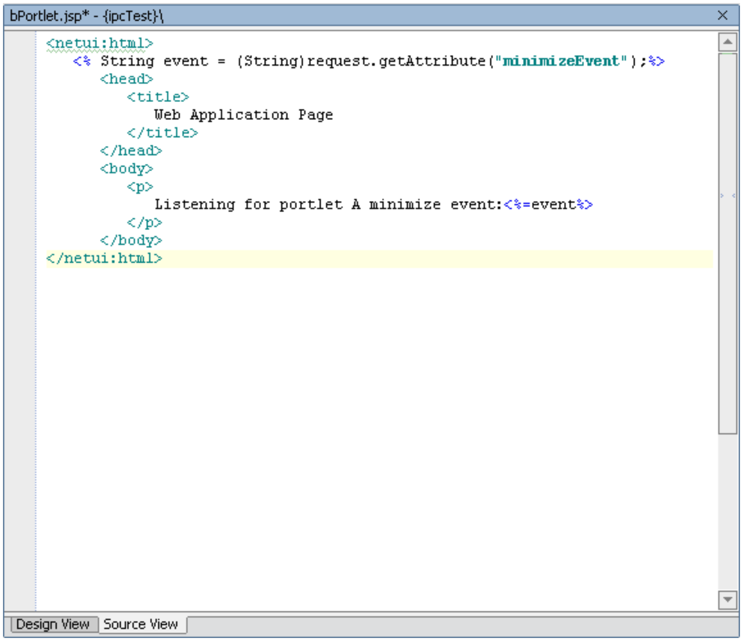
Listing 3-1 New JSP Code for bPortlet.jsp

```
<netui:html>
   <% String event = (String)request.getAttribute("minimizeEvent");%>
      <head>
         <title>
```

```
            Web Application Page
         </title>
      </head>
      <body>
         <p>
            Listening for portlet A minimize event:<%=event%>
         </p>
      </body>
</netui:html>
```

The source should look like example in Figure 3-5.

Figure 3-5 Updated JSP Source



```
<netui:html>
   <% String event = (String)request.getAttribute("minimizeEvent");%>
      <head>
         <title>
            Web Application Page
         </title>
      </head>
      <body>
         <p>
            Listening for portlet A minimize event:<%=event%>
         </p>
      </body>
</netui:html>
```

13. Save the file either by clicking the save button or by opening the File menu and selecting Save.

14. Repeat steps 6 and 7 to create a JSP portlet for bPortlet.

Create the Backing File

You now need to create the backing file that contains the instructions necessary to complete the communication between two portlets (for more information on backing files, please refer to Chapter 5, "Understanding Backing Files"). To create the backing file, do the following:

1. Expand the `WEB-INF` node and right-click `src` to open a context menu.
2. Select New>Folder.

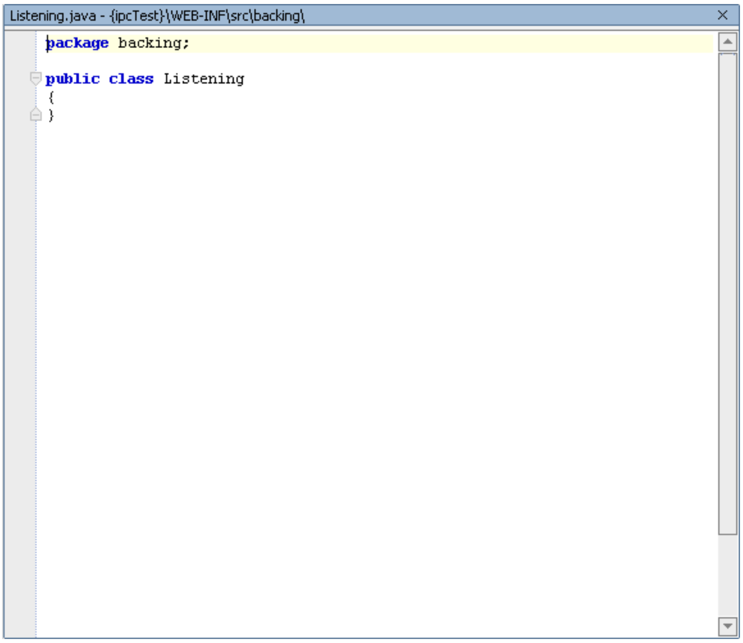
   The Create New Folder dialog box appears.
3. In Enter a new folder name, type `backing` and click OK.

   The folder `backing` will appear under `WEB-INF/src`.
4. Right-click backing and select New>Java Class.

   The New File dialog box appears.
5. In Name, enter `Listening.java` and click Create.

   The Source View of the new Java class appears (Figure 3-6).

Figure 3-6 Listening.java Source File



Listening.java - {ipcTest}\WEB-INF\src\backing\

```
package backing;

public class Listening
{
}
```

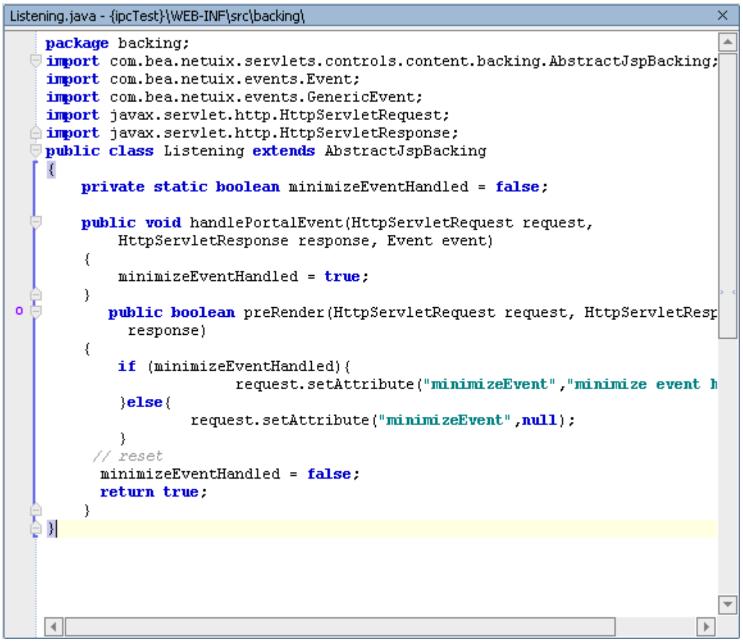
6. Copy the code from Listing 3-2 into `Listening.java`.

Listing 3-2 Backing File Code for Listening.java

```
package backing;

import com.bea.netuix.servlets.controls.content.backing.AbstractJspBacking;
import com.bea.netuix.events.Event;
import com.bea.netuix.events.GenericEvent;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class Listening extends AbstractJspBacking

{

    private static boolean minimizeEventHandled = false;

    public void handlePortalEvent(HttpServletRequest request,
        HttpServletResponse response, Event event)
    {
        minimizeEventHandled = true;
    }

       public boolean preRender(HttpServletRequest request, HttpServletResponse
         response)
    {
        if (minimizeEventHandled){

                request.setAttribute("minimizeEvent","minimize event handled");
        }else{
                request.setAttribute("minimizeEvent",null);
        }

     // reset
      minimizeEventHandled = false;
      return true;
    }

}
```

7. The source should now look like that shown in Figure 3-7.

Figure 3-7 listening.java with Updated Backing File Code



Listening.java - {ipcTest}\WEB-INF\src\backing\

```
package backing;
import com.bea.netuix.servlets.controls.content.backing.AbstractJspBacking;
import com.bea.netuix.events.Event;
import com.bea.netuix.events.GenericEvent;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class Listening extends AbstractJspBacking
{
    private static boolean minimizeEventHandled = false;

    public void handlePortalEvent(HttpServletRequest request,
        HttpServletResponse response, Event event)
    {
        minimizeEventHandled = true;
    }
       public boolean preRender(HttpServletRequest request, HttpServletResp
         response)
    {
        if (minimizeEventHandled){
                    request.setAttribute("minimizeEvent","minimize event h
        }else{
                request.setAttribute("minimizeEvent",null);
        }
     // reset
      minimizeEventHandled = false;
      return true;
    }
}
```

8. Save `Listening.java` either by opening the File menu and selecting Save or clicking the Save button.

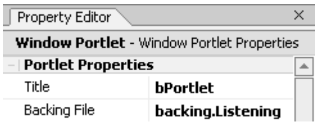
Attach the Backing File

Now you will attach the backing file created in Create the Backing File to bPortlet. Do the following:

1. In the Application tree, double-click `bPortlet.portlet` to open it.

2. In the Property Editor, under Portlet Properties, type `backing.Listening` into the Backing File field, as shown in Figure 3-8 and press Tab.

Figure 3-8 Attaching the Backing File in the Property Editor



Property Editor

Window Portlet - Window Portlet Properties

Portlet Properties	
Title	**bPortlet**
Backing File	**backing.Listening**

3. Save the file.

Add the Event Handler to bPortlet

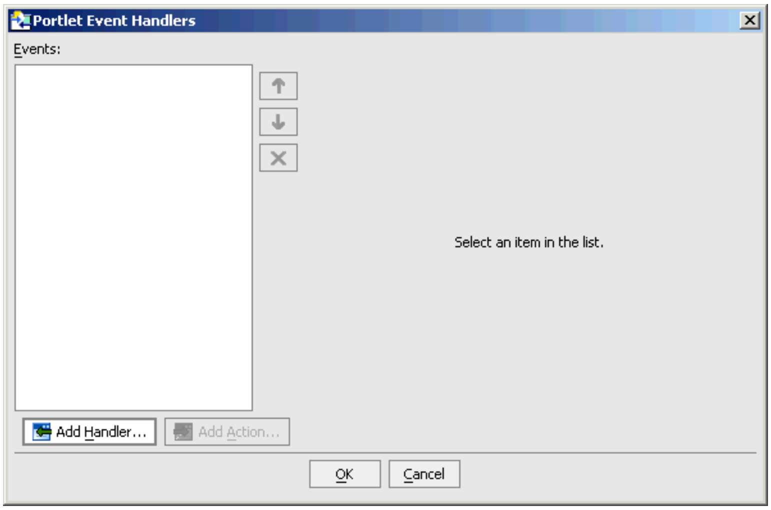
To add the event handler to bPortlet, do the following:

Note: bPortlet.portlet should be displayed in WebLogic Workshop. If it isn't, locate it under ipcTest in the application panel and double-click it.

1. In the Property Editor, click the ellipses button (...) next to Event Handlers.

   The Event Handler dialog box appears (Figure 3-9).

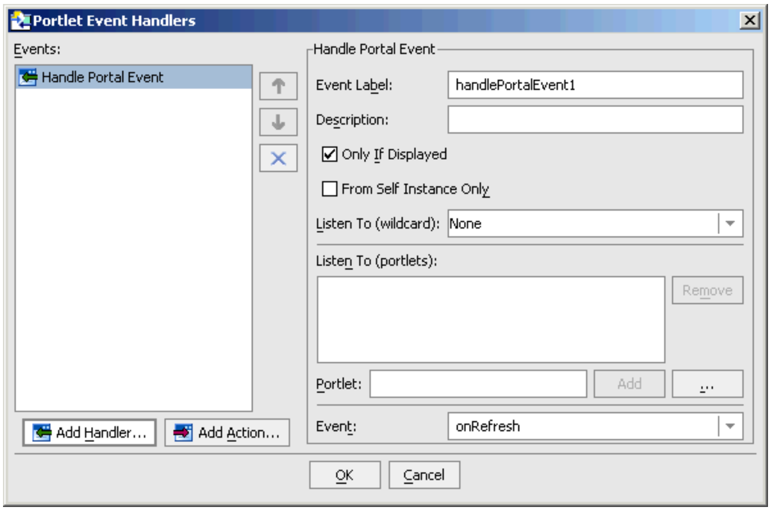
Figure 3-9 Event Handler Dialog Box



2. Click Add Handler to open the Event Handler drop-down list.
3. Select Handle Portal Event.

   The Event Handler dialog box expands to allow entry of more details (Figure 3-10).

Figure 3-10 Event Handler Dialog Box Expanded



Event Label: handlePortalEvent1

Description:

☑ Only If Displayed

☐ From Self Instance Only

Listen To (wildcard): None

Listen To (portlets):

Portlet:

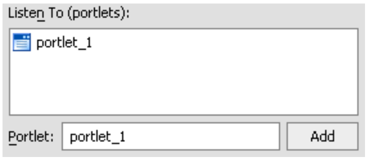
Event: onRefresh

4. Accept the defaults for all fields except Portlet.
5. In Portlet, click the ellipses button (...).

   The Open dialog box for ipcTest appears.
6. Double-click `aPortlet.portlet`.

   The Open dialog box closes and Portal_1 appears in the Listen to: list and the Portlet field (Figure 3-11). Portal_1 is the definition label of the portlet to which the event handler will listen.

Figure 3-11 Adding portlet_1



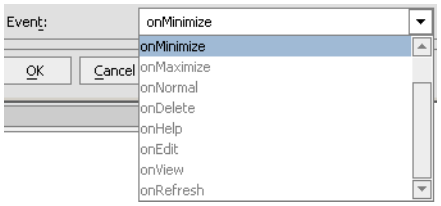
Listen To (portlets):

portlet_1

Portlet: portlet_1

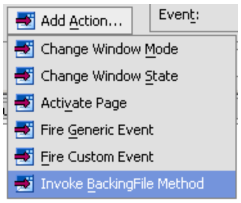
7. Click the Event drop-down control to open the list of portal event that the handler can listen for and select onMinimize, as shown in Figure 3-12.

Figure 3-12 Event Drop-down List



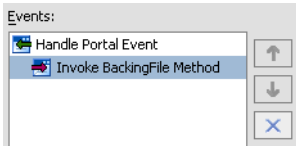
8. Click Add Action... to open the action drop-down list and select Invoke BackingFile Method, as shown in Figure 3-13.

Figure 3-13 Add Action Drop-down List



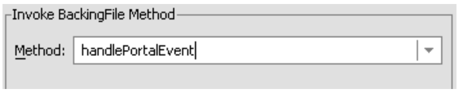
Invoke BackingFile Method appears on the Events list as a child to Handle Portal Event, as shown in Figure 3-14.

Figure 3-14 Event List with Action Added



9. In Method, select pr enter `handlePortalEvent` (Figure 3-15).

Figure 3-15 Adding the Backing File Method



10. Click OK.

    The event handler is added. Note that the Event Handler field in the Property Editor now reads “1 Event Handler.”

11. Save the file.

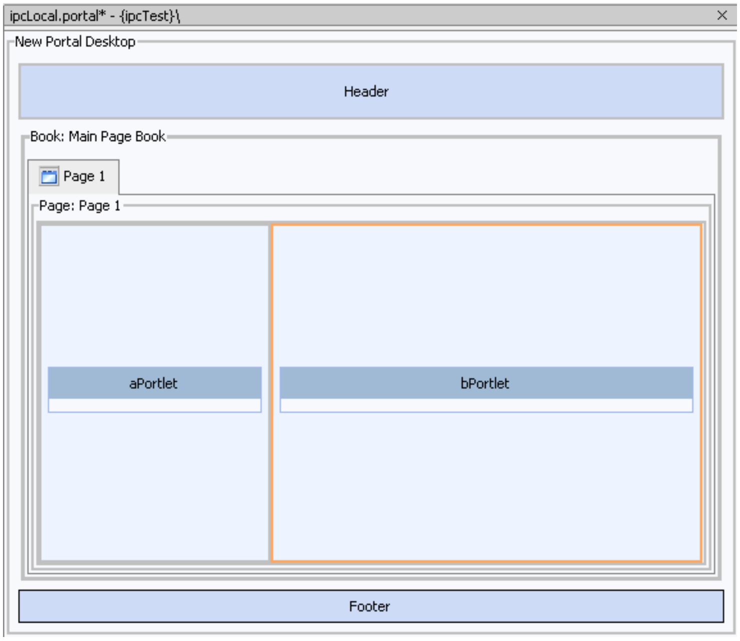
Step 2: Test the Application

To test the application, do the following:

1. Create a portal called `ipcLocal.portal` by doing the following:
   a. Right-click ipcTest and select New>Portal.
   b. In the New File dialog box's File Name field, enter ipcLocal.
   c. Click Create.

   When the portal is successfully created, its layout will appear in WebLogic Workshop.
2. Drag both aPortlet and bPortlet from the Data Palette onto the portal layout, as shown in Figure 3-16.

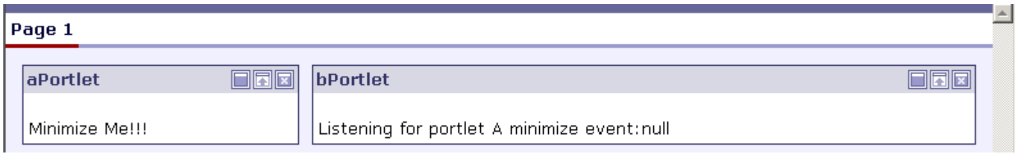
Figure 3-16 Portal Layout with Portlets Added



3. Save the portal either by clicking the save button or opening the file menu and selecting Save.
4. Open the Portal Menu and select Open Current Portal...

The portal will render in your browser (Figure 3-17).

Figure 3-17 ipcLocal Portal in Browser



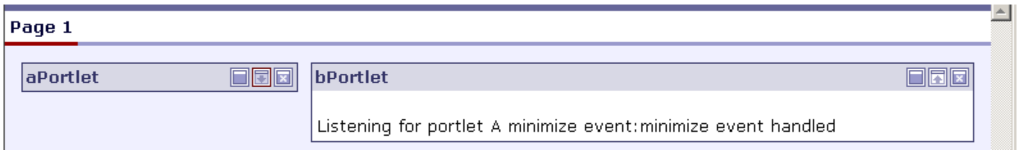
Minimize Me!!!

Listening for portlet A minimize event:null

5. Minimize aPortlet.

   Note the content change in bPortlet (Figure 3-18).

Figure 3-18 ipcLocal Portal with aPortlet Minimized



Listening for portlet A minimize event:minimize event handled

Summary

In this exercise, you added to the portal application components created in Chapter 2, "Setting Up the Examples" two JSP portlets. One portlet, aPortlet, was fairly simple, while the second portlet, bPortlet, surfaced a more complex JSP file, leveraged a backing file, and contained a portal event handler. When you tested the application, you observed how the two portlets communicated when an event occurred on aPortlet. This is called local interportlet communications.

Establishing IPC by Using Custom and Page Flow Events

In Chapter 3, “A Simple Example of Establishing IPC,” we showed a very basic example of establishing interportlet communications by using a portal framework event. Usually, you will want to provide more complex IPC in your applications; for example, you might want to handle an event that is not previously defined in WebLogic Portal, called a “custom” event, or one that is the result of a page flow action, or a “page flow” event. WebLogic Portal 8.1 with Service Pack 4 and later provide event handlers to accomplish both of these types of interportlet communications.

What is a Custom Event?

A custom event is simply an event you, as the developer, define to the application; in other words, it is not an event provided out-of-the-box with WebLogic Portal 8.1. A custom event can pass a developer-defined payload or fire any other predefined action and can be fired declaratively or based on a methods called in a backing file. A user will be able to specify that an event should be handled by a method in a backing file.

What is a Page Flow Event?

A page flow event is any occurrence during a page flow lifecycle that can trigger an action on another portlet. For example, if a user submits authentication information via a login portlet that uses a page flow, submission of the login form thereon can signal other portlets listening to it to query various databases and return information specific to the authenticated user and specific to the listening portlets.

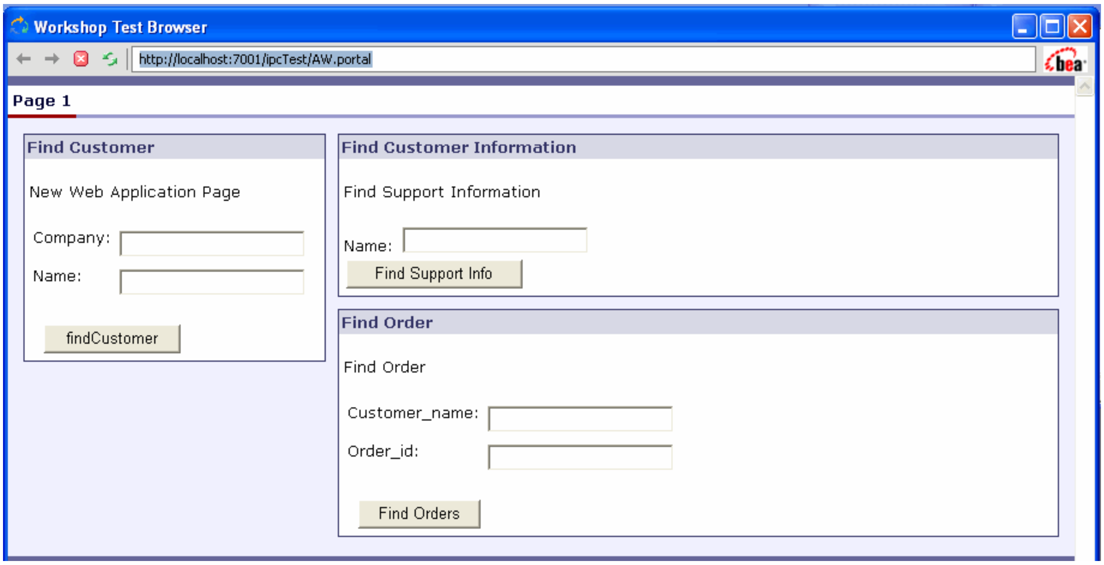
Establishing Interportlet Communications by Using Custom and Page Flow Event Handlers: An Example

The following exercise will create a portal desktop that uses both custom and page flow events to achieve interportlet communications. When completed, the portal will contain three portlets:

- A *Find Customer lookup portlet* into which users will enter the name of a customer.
- A *Find Customer Information portlet*, which displays information about the customer whose name was submitted in the customer lookup portlet
- An *Find Orders portlet*, which displays order information for the customer whose name was submitted in the customer lookup portlet.

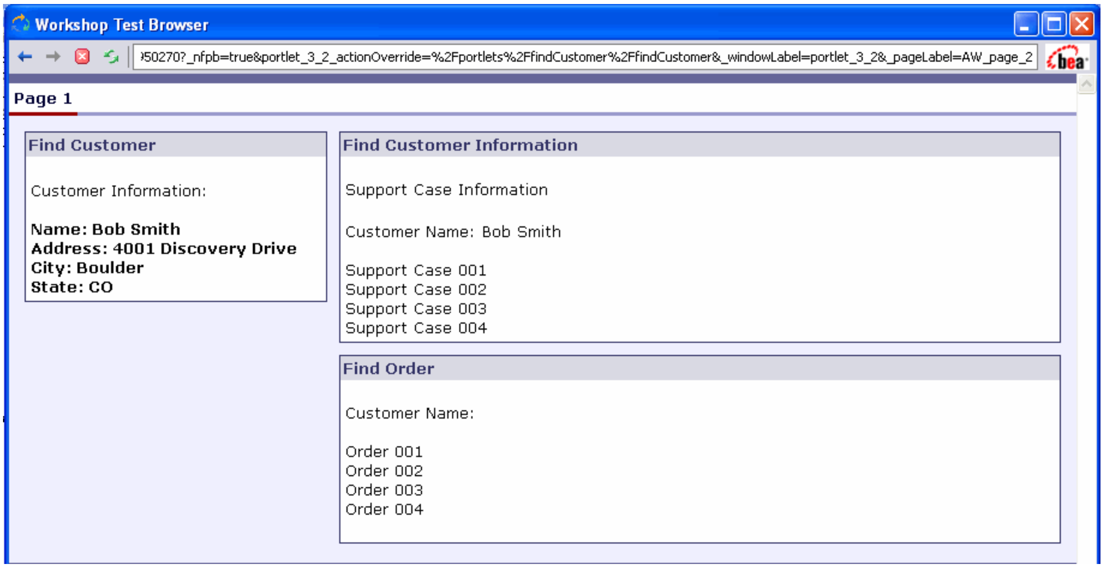
When initially rendered, the completed portal will look like the example in Figure 4-1.

Figure 4-1 Portal as Initially Rendered



Once the customer name is submitted from Customer Lookup portlet, the portal will look like the example in Figure 4-2

Figure 4-2 Portal with Customer Name Submitted


Workshop Test Browser
⁾50270?_nfpb=true&portlet_3_2_actionOverride=%2Fportlets%2FfindCustomer%2FfindCustomer&_windowLabel=portlet_3_2&_pageLabel=AW_page_2
Page 1

Find Customer

Customer Information:

Name: Bob Smith
Address: 4001 Discovery Drive
City: Boulder
State: CO

Find Customer Information

Support Case Information

Customer Name: Bob Smith

Support Case 001
Support Case 002
Support Case 003
Support Case 004

Find Order

Customer Name:

Order 001
Order 002
Order 003
Order 004

Creating this example requires these steps:

- Step 1: Create the File Structure
- Step 2: Create the Backing Files
- Step 3: Create the Customer Lookup Results Portlet
- Step 4: Create the Orders Lookup Portlet
- Step 5: Create the Customer Lookup Portlet
- Step 6: Create and Populate the Portal File
- Step 7: Test the Portal

How this Example Achieves IPC

All three portlets in this exercise use backing files to fire actions that establish and execute the interportlet communication between them. When a backing file is attached to a portlet, the action Invoke BackingFile Method is made available on the Event Handler tool. After you select the Invoke BackingFile Method action, you select the appropriate method from the backing file from the Method drop-down list. When the portlet detects the custom event for which it is listening, it invokes the attached backing file and executes the specified method. Backing files can have multiple methods, any of which are made available to the event handler.

Backing files are Java classes that implement the `com.bea.netuix.servlets.controls.content.backing.JspBacking` interface or extend the `com.bea.netuix.servlets.controls.content.backing.AbstractJspBacking interface` abstract class. They allow you to you to programatically add functionality to a portlet, enabling preprocessing (for example, authentication) prior to rendering the portal controls. Before attempting this exercise, we recommend that you refer to Chapter 5, "Understanding Backing Files."

Before You Begin

If you have not set up your development environment as described in Chapter 2, "Setting Up the Examples," please refer to that section and follow these steps:

- Step 1: Create the Domain
- Step 2: Create the Enterprise Application
- Step 3: Create the Portal Web Application (Project)

After completing these steps, you will have portal domain called ipcDomain and a portal application called ipcTest.

Note on Entering Text in Dialog Boxes

Because so much of the text you will enter in this exercise refers to case-sensitive names in the various JPF, JSP, and backing files used to establish IPC, please enter all text *exactly* as it appears in the procedure.

Step 1: Create the File Structure

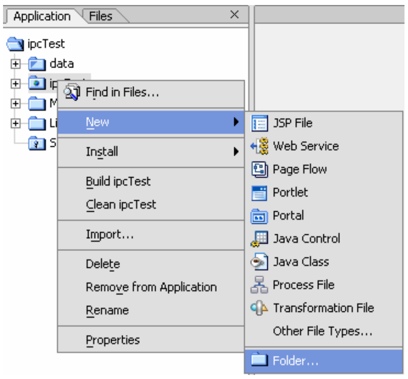
Next, you need to create some additional folders on the file tree. These directories will contain the portlets and their components (for example JSPs and page flow files) necessary for successfully establishing IPC.

Note: While creating separate folders for each portal is optional, to do so is a best practice as it allows for better organization of the application and reduces clutter on the application root.

To create the folders necessary for the application, use this procedure:

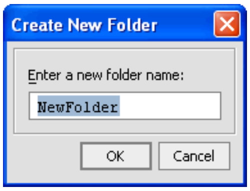
1. Right-click the project name (ipcTest) and select **New**>**Folder** (Figure 4-3).

Figure 4-3 Creating a New Folder



The New Folder dialog box appears (Figure 4-4).

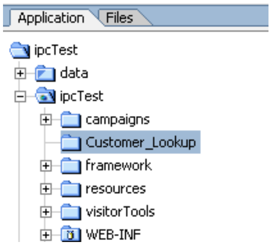
Figure 4-4 New Folder Dialog Box



2. In **Enter new folder name:**, type `Customer_Lookup`; click **OK**.

   The Customer_Lookup folder will appear in the file tree, at the project root (Figure 4-5):

Figure 4-5 File Tree with Customer_Lookup Folder Added

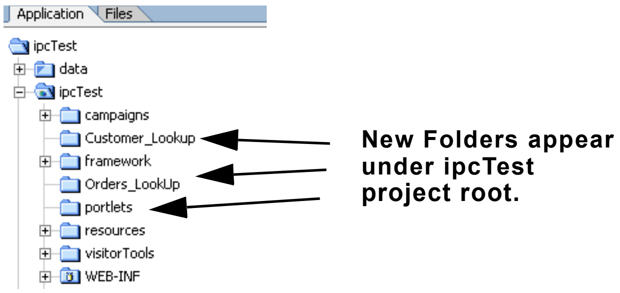


3. Repeat steps 1 and 2 to create two more folders. Name these folders:
   – Orders_LookUp (note the use of capitalization in “LookUp”; you must type this word exactly as shown here)

– portlets

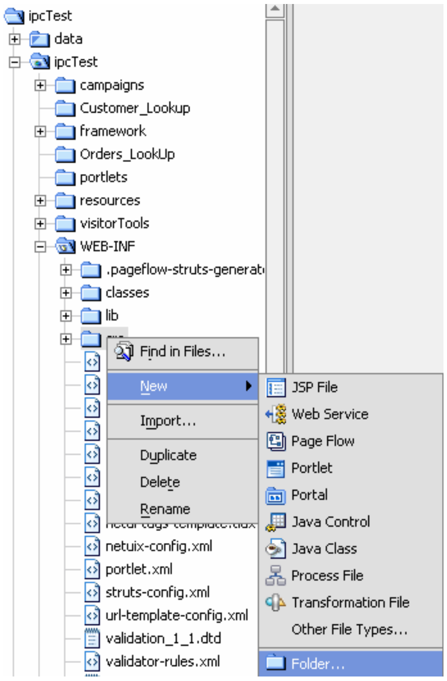
When you are done creating the folders, the project root should look like the example in Figure 4-6.

Figure 4-6 FIle Tree with New Folders Added



4. Next, create the `backing` folder. This folder will store the backing files you will create in a few moments. To do so, open the `ipcTest/WEB-INF/src` folder and repeat steps 1 and 2 (Figure 4-7):

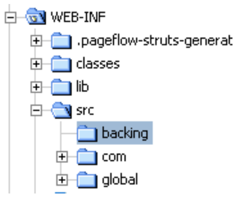
Figure 4-7 Opening the ipcTest/WEB-INF/src Folder



5. In the New Folder dialog box, enter `backing` and click **OK**.

   The backing file will appear under `WEB-INF/src` in the file tree (Figure 4-8):

Figure 4-8 File Tree with backing Folder Added



Summary

With the completion of step 2, you have added all the new folders required by the application. Note that when you create the page flows in later steps, an additional folder for each page flow will be created under the folder that contains the page flow.

Step 2: Create the Backing Files

A backing file is a simple Java class implementing the JspBacking interface. Backing files work in conjunction with JSPs. The JSPs allow you to code the presentation logic, while the backing files allow you to code simple business logic. Backing files are always run before the JSPs. A backing file has a lifecycle with four methods that are run in order on all objects. You can effect the underlying object from the BackingContext. The BackingContext should be used from the backing file and the PresentationContext should be used from the JSPs. As described earlier, backing files should be stored in `WEB-INF/src/backing`.

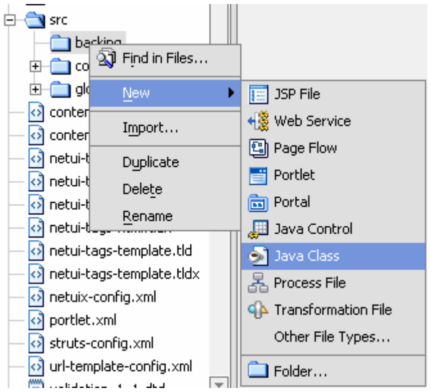
In this step, you will create three backing files, one for each portlet in the application. To save time creating these files, we have provided the necessary code in this document. If you are reading this online, you can simply cut and paste the code into the appropriate empty file. If you are using a hardcopy or PDF version of this document and cannot obtain an online (HTML) copy, you can type the code *exactly as it appears* in the code samples.

To create the backing files for this exercise, use this procedure:

1. First create `FindCustomerBacking.java`:

   a. Right-click `WEB-INF/src/backing` and select New>Java Class:

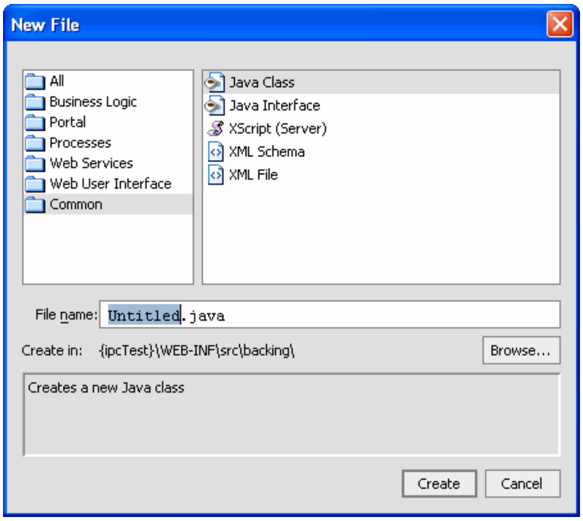
Figure 4-9 Creating a Backing File



The New File dialog box appears (Figure 4-10).

Note: Common should be selected in the left pane and Java Class in the right pane. If not, select them now.

Figure 4-10 New File Dialog Box; Common and Java Class Selected



File name: Untitled.java

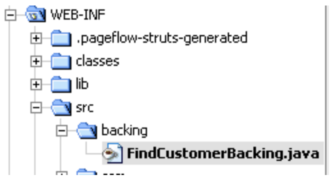
Create in: {ipcTest}\WEB-INF\src\backing\

Creates a new Java class

b. In File name:, type `FindCustomerBacking` and click Create.

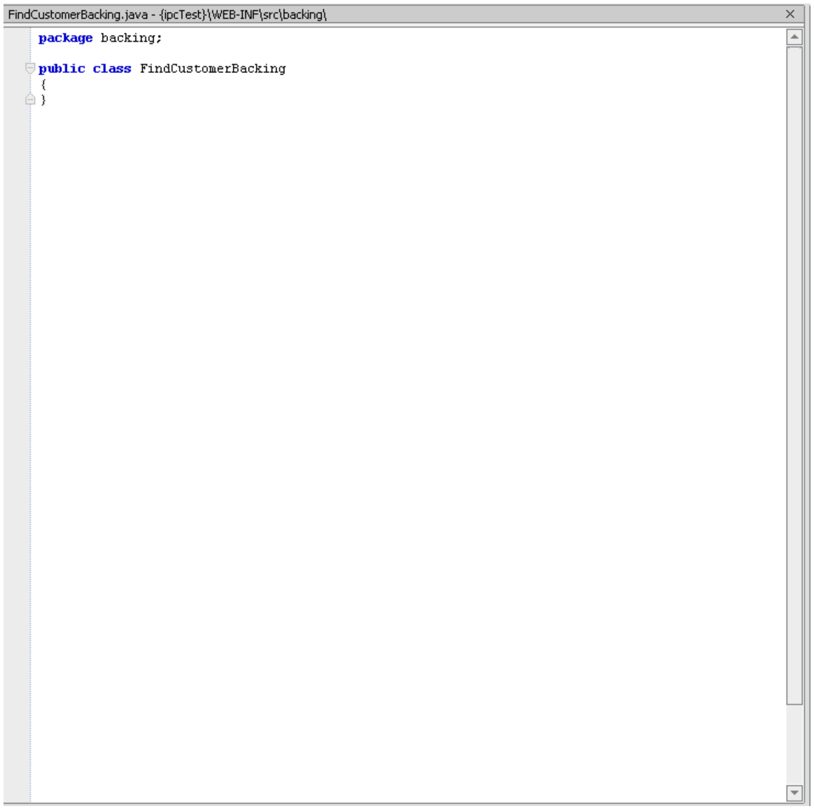
`FindCustomerBacking.java` appears under `WEB-INF/src/backing` (Figure 4-11):

Figure 4-11 FindCustomerBacking.java under WEB-INF/src/backing



and a file editor opens in WebLogic Workshop (Figure 4-12).

Figure 4-12 Backing File Template in the File Editor



```
package backing;

public class FindCustomerBacking
{
}
```

c. Delete *all* content in the file editor and replace it with the entire code in Listing 4-1.

Listing 4-1 FindCustomerBacking.java Sample Code

```
package backing;

import com.bea.netuix.servlets.controls.content.backing.JspBacking;
import com.bea.netuix.servlets.controls.content.backing.AbstractJspBacking;
import com.bea.netuix.events.Event;
import com.bea.netuix.events.GenericEvent;
import com.bea.netuix.events.PageFlowEvent;
import com.bea.netuix.servlets.controls.portlet.backing.PortletBackingContext;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class FindCustomerBacking extends AbstractJspBacking
{
    public void foundCustomer(HttpServletRequest request, HttpServletResponse
       response, Event event)
   {
        // Get result from form post and inject inside remote IPC

        PortletBackingContext context =
          PortletBackingContext.getPortletBackingContext(request);
        String message = request.getParameter(context.getInstanceLabel() +
          "{actionForm.name}");
        context.fireCustomEvent("myCustomEvent", message);
    }
}
```

   d. Save the file by either clicking the Save icon or by opening the File menu and selecting Save.

2. Next, create `ListenCustomerName.java`:

   a. Repeat steps a through c in step 1, above; however, in File name (step b) enter `ListenCustomerName`.

   b. Click Create.

      The file is created (it will appear under `WEB-INF/src/backing` in the file tree) and a backing file template appears in the file editor.

   c. Delete *all* existing code in the file editor and replace it with the *entire* code in Listing 4-2.

Listing 4-2 ListenCustomerName.java Code Sample

```
package backing;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import com.bea.netuix.events.Event;
import com.bea.netuix.events.CustomEvent;
import com.bea.netuix.servlets.controls.content.backing.AbstractJspBacking;

public class ListenCustomerName extends AbstractJspBacking
```

```
{
    public void listenCustomerName( HttpServletRequest request,
       HttpServletResponse response, Event event)
    {

        CustomEvent customEvent = (CustomEvent) event;

        String message = (String) customEvent.getPayload();

        HttpSession mySession = request.getSession();

        mySession.setAttribute("customerName", message);

    }

}
```

d. Save the file by either clicking the Save icon or by opening the File menu and selection Save.

3. Finally, create `ReceiveCustomerInfo.java`:

a. Repeat steps a through c in step 1, above; however, in File name (step b) enter `ReceiveCustomerInfo`.

b. Click Create.

The file is created and a backing file template appears in the file editor

c. Delete *all* existing code in the file editor and replace it with the *entire* code in Listing 4-2.

Listing 4-3 ReceiveCustomerInfo.java Code Sample

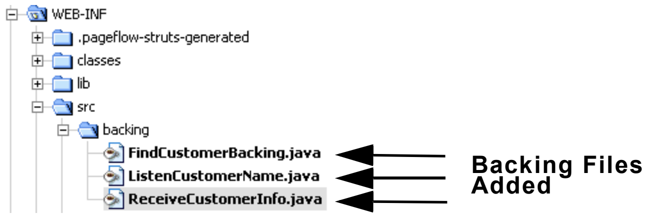
```
package backing;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import com.bea.netuix.events.Event;
import com.bea.netuix.events.CustomEvent;
import com.bea.netuix.servlets.controls.content.backing.AbstractJspBacking;

public class ReceiveCustomerInfo  extends AbstractJspBacking
{
    public void listenCustomerName( HttpServletRequest request,
        HttpServletResponse response, Event event)
    {

        CustomEvent customEvent = (CustomEvent) event;

        String message = (String) customEvent.getPayload();

        HttpSession mySession = request.getSession();

        mySession.setAttribute("customerName", message);

    }

}
```

d. Save the file by either clicking the Save icon or by opening the File menu and selection Save.

Summary

With the completion of Step 3, you have created the necessary backing files and have stored them in the `WEB-INF/src/backing` folder. That folder should look like the example in Figure 4-13.

Figure 4-13 WEB-INF/src/backing with All Backing Files Created



Step 3: Create the Customer Lookup Results Portlet

In this step, you will create the first of the three portlets. This portlet will listen to the customer lookup portlet and when it hears a specific event (in this case, a customer name being submitted), will fire an action in response (in this case, return information about the requested customer).

To create this portlet you will create four files: one page flow file, two JSP files, and the `.portlet` file. The `.portlet` file is simply a container for the other three files; however, this is the file upon which critical properties are set, specifically the backing file and the event handlers.

Note: To save you time and to reduce the chance of entry errors when creating these files, we have provided the necessary code in this document. If you are reading this online (HTML), you can simply cut and paste the code into the appropriate empty file. If you are using a hardcopy or PDF version of this document and cannot obtain an online copy, you can type the code *exactly as it appears* in the code samples.

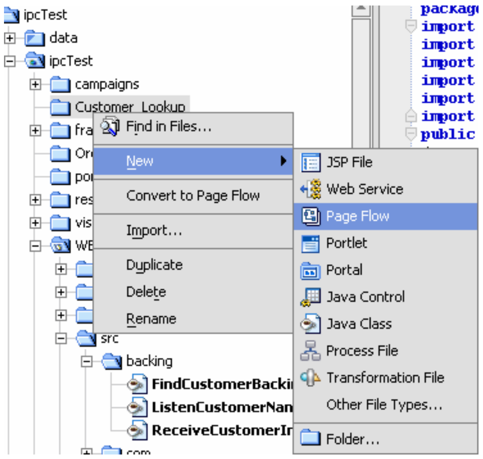
Create the Page Flow (FindCustomerController.jpf)

The FindCustomer page flow (`FindCustomerController.jpf`) establishes the order in which pages are rendered in the Customer Lookup Results portlet. This page flow executes after a name is submitted on the Customer Lookup Portlet.

To create the FindCustomer page flow, use this procedure (the procedure assumes that WebLogic Workshop is running):

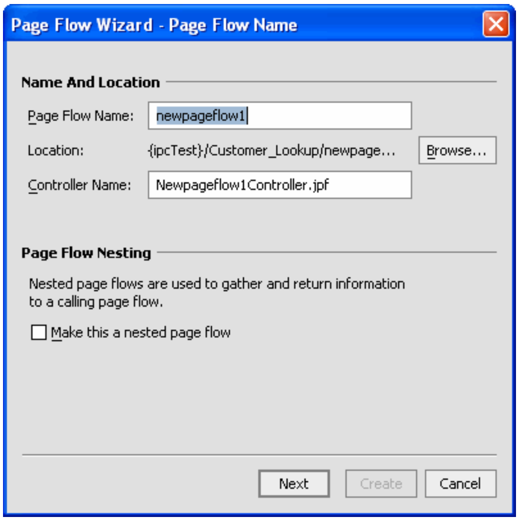
1. On the file tree, right-click Customer_Lookup and select New > Page Flow (Figure 4-14).

Figure 4-14 New>Page Flow Selected



The Page Flow Wizard appears (Figure 4-15).

Figure 4-15 Page Flow Wizard

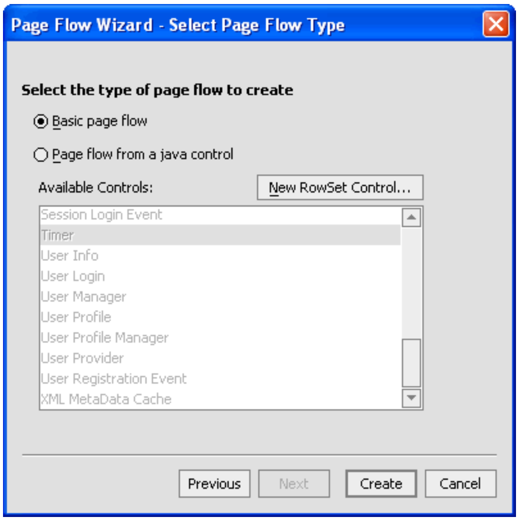

Page Flow Wizard - Page Flow Name
Name And Location
Page Flow Name: newpageflow1
Location: {ipcTest}/Customer_Lookup/newpage...
Browse...
Controller Name: Newpageflow1Controller.jpf
Page Flow Nesting
Nested page flows are used to gather and return information to a calling page flow.
Make this a nested page flow
Next
Create
Cancel

2. In Page Flow Name, enter findCustomer; note that the same text replaces the string Newpageflow1 in the Controller Name field, so that this field reads "FindCustomerController" (the lowercase "f" in Page Flow Name is translated to a capital "F" in Controller Name).

3. Click Next.

   The Select Page Flow Type dialog box appears (Figure 4-16).

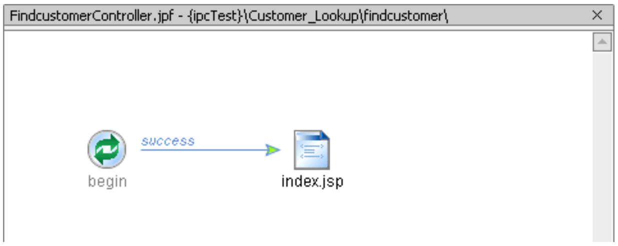
Figure 4-16 Select Page Flow Type Dialog Box



4. Ensure that Basic page flow is selected and click Create.

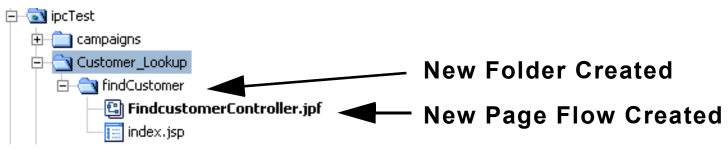
   A basic page flow design appears in the Page Flow Designer (Figure 4-17):

Figure 4-17 FindCustomerController.jpf in Page Flow Designer — Flow View


FindcustomerController.jpf - {ipcTest}\Customer_Lookup\findcustomer\
success
begin
index.jsp

and `FindCustomerController.jpf` (Figure 4-18) appears in the file tree. Note that when you create this page flow, a new folder bearing the page flow name (findCustomer) appears in the `Customer_Lookup` folder (Figure 4-18) and an additional JSP, `index.jsp` is created. The `findCustomer` folder will host the files that comprise this portlet and `index.jsp` will serve as a template for the `result.jsp` file.

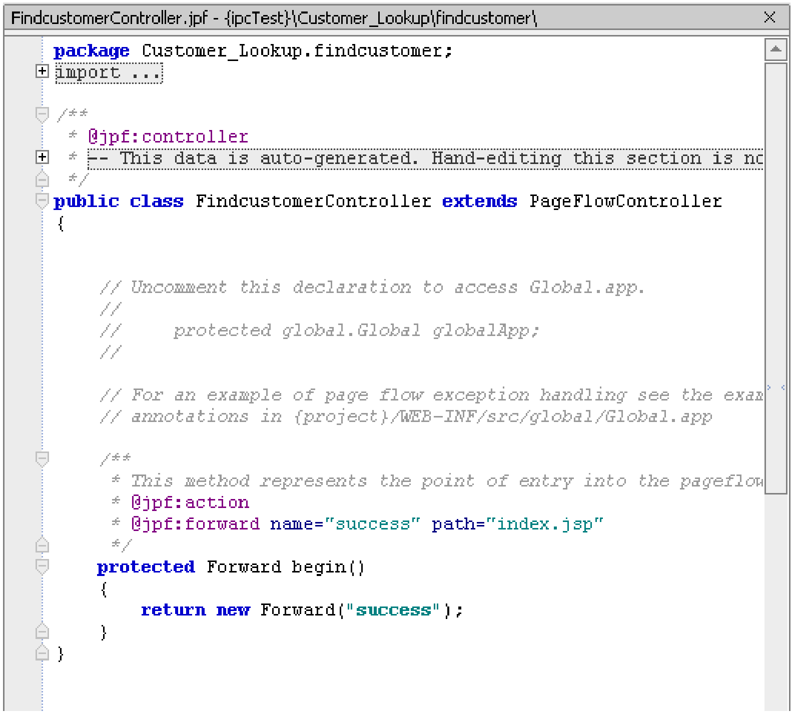
Figure 4-18 FindCustomerController.jpf in findCustomer Folder



5. At the bottom of the Page Flow Designer, select Source View.

   The view in the Page Flow Designer changes to that shown in Figure 4-19.

Figure 4-19 Default Page Flow Code in Source View


FindcustomerController.jpf - {ipcTest}\Customer_Lookup\findcustomer\

```
package Customer_Lookup.findcustomer;
import ...

/**
 * @jpf:controller
 * -- This data is auto-generated. Hand-editing this section is nc
 */
public class FindcustomerController extends PageFlowController
{


    // Uncomment this declaration to access Global.app.
    //
    //     protected global.Global globalApp;
    //

    // For an example of page flow exception handling see the exa
    // annotations in {project}/WEB-INF/src/global/Global.app

    /**
     * This method represents the point of entry into the pageflow
     * @jpf:action
     * @jpf:forward name="success" path="index.jsp"
     */
    protected Forward begin()
    {
        return new Forward("success");
    }
}
```

6. Highlight the entire contents of the Designer and delete it, so that the Source View is empty.
7. Copy the code sample in Listing 4-4 and paste it into the empty Page Flow Designer Source View.

Listing 4-4 FindCustomerController.jpf Code Sample

```
package Customer_Lookup.findCustomer;
import com.bea.wlw.netui.pageflow.FormData;
import com.bea.wlw.netui.pageflow.Forward;
import com.bea.wlw.netui.pageflow.PageFlowController;


/**
 * @jpf:controller
 * @jpf:view-properties view-properties::
 * <!-- This data is auto-generated. Hand-editing this section is not
 * recommended. -->
 * <view-properties>
 * <pageflow-object id="pageflow:/Customer_Lookup/findCustomer/
 * FindCustomerController.jpf"/>
 * <pageflow-object id="action:begin.do">
 *   <property value="80" name="x"/>
 *   <property value="100" name="y"/>
 * </pageflow-object>
 * <pageflow-object id="action:findCustomerInfo.do#Customer_Lookup.
 * findCustomer.FindCustomerController.FindCustomerInfoForm">
 *   <property value="680" name="x"/>
 *   <property value="100" name="y"/>
 * </pageflow-object>
 * <pageflow-object id="action-call:@page:findCustomer.jsp@#@action:
 * findCustomerInfo.do#Customer_Lookup.findCustomer.FindCustomerController.
 * FindCustomerInfoForm@">
 *   <property value="276,460,460,644" name="elbowsX"/>
 *   <property value="92,92,92,92" name="elbowsY"/>
 *   <property value="East_1" name="fromPort"/>
 *   <property value="West_1" name="toPort"/>
 * </pageflow-object>
 * <pageflow-object id="page:findCustomer.jsp">
 *   <property value="240" name="x"/>
 *   <property value="100" name="y"/>
 * </pageflow-object>
 * <pageflow-object id="page:result.jsp">
 *   <property value="560" name="x"/>
 *   <property value="100" name="y"/>
 * </pageflow-object>
 * <pageflow-object id="forward:path#success#findCustomer.
 * jsp#@action:begin.do@">
 *   <property value="116,160,160,204" name="elbowsX"/>
 *   <property value="92,92,92,92" name="elbowsY"/>
 *   <property value="East_1" name="fromPort"/>
 *   <property value="West_1" name="toPort"/>
 *   <property value="success" name="label"/>
 * </pageflow-object>
 * <pageflow-object
 * id="forward:path#success#result.jsp#@action:
 * findCustomerInfo.do#Customer_Lookup.findCustomer.FindCustomerController.
 * FindCustomerInfoForm@">
 *   <property value="644,620,620,596" name="elbowsX"/>
 *   <property value="81,81,81,81" name="elbowsY"/>
 *   <property value="West_0" name="fromPort"/>
 *   <property value="East_0" name="toPort"/>
 *   <property value="success" name="label"/>
 * </pageflow-object>
 * <pageflow-object id="formbeanprop:Customer_Lookup.findCustomer.
 *    FindCustomerController.FindCustomerInfoForm#name#java.lang.String"/>

 * <pageflow-object id="formbean:Customer_Lookup.findCustomer.
 * FindCustomerController.FindCustomerInfoForm"/>

 * <pageflow-object id="action-call:@page:result.jsp@#@action:
 * findCustomerInfo.do#Customer_Lookup.findCustomer.FindCustomerController.
 * FindCustomerInfoForm@">
 *   <property value="596,620,620,644" name="elbowsX"/>
 *   <property value="92,92,81,81" name="elbowsY"/>
 *   <property value="East_1" name="fromPort"/>
 *   <property value="West_0" name="toPort"/>
 * </pageflow-object>
 * </view-properties>
 * ::
 */

public class FindCustomerController extends PageFlowController
{

    // Uncomment this declaration to access Global.app.
    //
    //     protected global.Global globalApp;
    //



    // For an example of page flow exception handling see the example "catch"
and "exception-handler"
   // annotations in {project}/WEB-INF/src/global/Global.app

    /**
     * This method represents the point of entry into the pageflow
     * @jpf:action
     * @jpf:forward name="success" path="findCustomer.jsp"
     */
    protected Forward begin()
    {
        return new Forward("success");
    }

    /**
     * @jpf:action
     * @jpf:forward name="success" path="result.jsp"
     */

    protected Forward findCustomerInfo(FindCustomerInfoForm form)
    {
        return new Forward("success");
    }

    /**
     * FormData get and set methods may be overwritten by the Form Bean editor.
     */

    public static class FindCustomerInfoForm extends FormData
    {
        private String name;
        public void setName(String name)
        {
            this.name = name;
        }

        public String getName()
        {
            return this.name;
        }
    }
}
```

Note: You might see two “File could not be found” warnings (indicated by a green squiggle) in the Page Flow Designer. You can ignore them, as they refer to files that you will create in the next two steps.

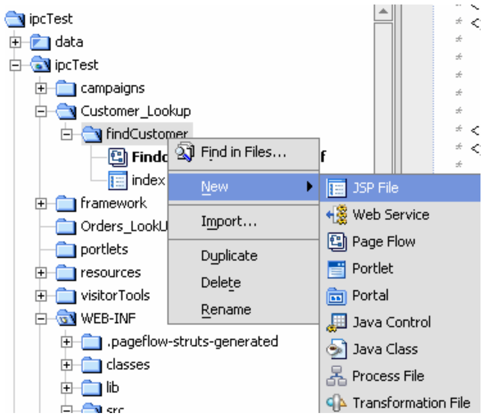
8. Save the file either by clicking the Save icon or by opening the File menu and selecting Save.

Create findCustomer.jsp

To create `findCustomer.jsp`, use this procedure:

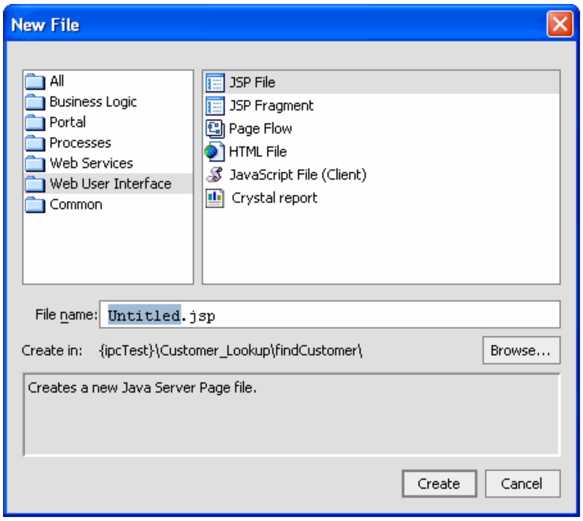
1. Right-click findCustomer in the file tree and select New > JSP file (Figure 4-20).

Figure 4-20 Selecting New>JSP for findCustomer



The New File dialog box appears (Figure 4-21).

Figure 4-21 New File Dialog Box; JSP File Selected



File name: Untitled.jsp

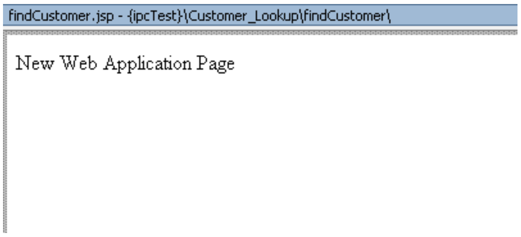
Create in: {ipcTest}\Customer_Lookup\findCustomer\

Creates a new Java Server Page file.

2. Ensure that Web User Interface and JSP File are selected; if not, do so now.
3. In File Name, type findCustomer and click Create.

The file is created; a new JSP template will appear in the JSP Designer (Figure 4-22):

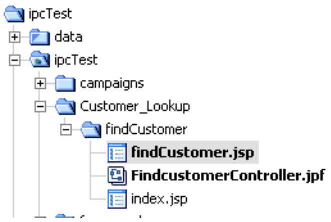
Figure 4-22 New JSP Template


findCustomer.jsp - {ipcTest}\Customer_Lookup\findCustomer\

New Web Application Page

and the new file appears in the `findCustomer` folder (Figure 4-23).

Figure 4-23 findCustomer.jsp File in findCustomer Folder



4. At the bottom of the JSP Designer, click Source View to display the JSP code.

5. Select the entire listing and delete it.

6. Copy the code in Listing 4-5 and paste it into the empty JSP Designer.

Listing 4-5 findCustomer.jsp Sample Code

```
<%@ page language="java" contentType="text/html;charset=UTF-8"%>
<%@ taglib uri="netui-tags-databinding.tld" prefix="netui-data"%>
<%@ taglib uri="netui-tags-html.tld" prefix="netui"%>
<%@ taglib uri="netui-tags-template.tld" prefix="netui-template"%>

<netui:html>


    <body>
        <p> Find Support Information </p>
        <netui:form action="findCustomerInfo">
            <div>
                Name:
```

```
                <netui:textBox dataSource="{actionForm.name}" defaultValue=
                    ""></netui:textBox>
                <br/>
            </div>
            <netui:button value="Find Support Info" type="submit"/>
        </netui:form>
    </body>
</netui:html>
```

7. Save the file either by clicking the Save icon or by opening the File menu and selecting Save.

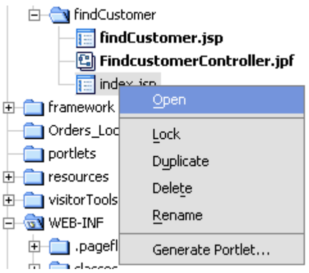
Create result.jsp

`result.jsp` contains the presentation logic for the final page of the successful execution of the findCustomer page flow (`findCustomerController.jpf`). This page renders after the name submitted in the Customer Lookup portlet is authenticated and will present specified customer data.

To create `result.jsp`, use this procedure.

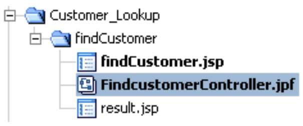
1. Locate and right-click `index.jsp` under `Customer_Lookup/findCustomer` to open the file's context menu (Figure 4-24).

Figure 4-24 `index.jsp` Context Menu



2. Select Rename to open the file rename function and replace `index.jsp` with `result.jsp` (Figure 4-25).

Figure 4-25 `result.jsp` Replaces `index.jsp` in File Tree



3. In the JSP Designer, select Source View to display the JSP code.
4. Highlight the entire code listing and delete it.
5. Copy the code listing in Listing 4-6 and paste it into the JSP designer.

Listing 4-6 Code Sample for result.jsp

```
<%@ page language="java" contentType="text/html;charset=UTF-8"%>
<%@ taglib uri="netui-tags-databinding.tld" prefix="netui-data"%>
<%@ taglib uri="netui-tags-html.tld" prefix="netui"%>
<%@ taglib uri="netui-tags-template.tld" prefix="netui-template"%>
<netui:html>

    <body>
        <p>
            Support Case Information
        </p>

        Customer Name:

        <%=session.getAttribute("customerName")
        %>

       <%
        session.setAttribute("customerName", "");
       %>

        <netui:form action="findCustomerInfo">
            <netui:label value="{actionForm.name}"/>
        </netui:form>

            <div>

         <br>

                Support Case 001<br/>
                Support Case 002<br/>
                Support Case 003<br/>
                Support Case 004<br/>
            </div>

        </body>
</netui:html>
```

---

6. Save the file either by clicking the Save icon or by opening the File menu and selecting Save.

Note: If you reopen `FindCustomerController.jpf` after saving both JSPs, you will see that the warnings described in the note following Listing 4-4 do not appear.

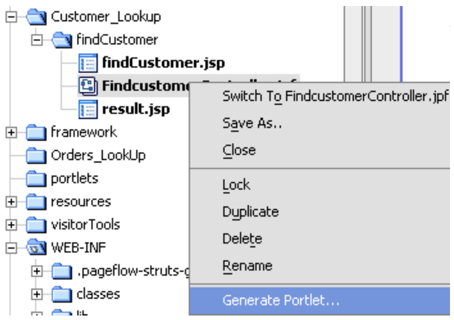
Create the Customer Lookup Portlet

The Customer Lookup portlet is the container for the page flow and the JSPs that you just created. This is the file to which you'll attach the backing file and implement the event handlers necessary to achieve interportlet communications.

To create the Customer Lookup portlet, use this procedure.

1. Under Customer_Lookup/findCustomer, right-click FindCustomerController.jsp to open the file's context menu (Figure 4-26).

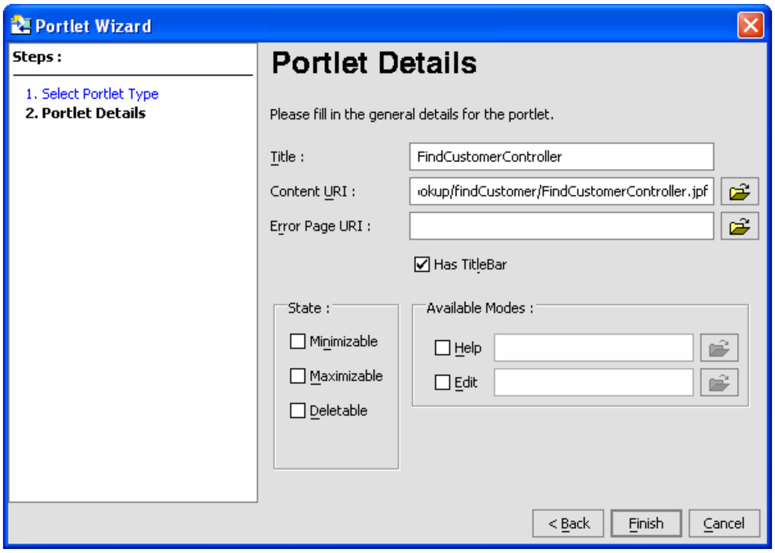
Figure 4-26 `FindCustomerController.jpf` Context Menu



2. Select Generate Portlet

   The Portlet Details dialog box appears (Figure 4-27).

Figure 4-27 Portlet Details Dialog Box



Portlet Details

Please fill in the general details for the portlet.

Title :	FindCustomerController
Content URI :	okup/findCustomer/FindCustomerController.jpf
Error Page URI :	

☑ Has TitleBar

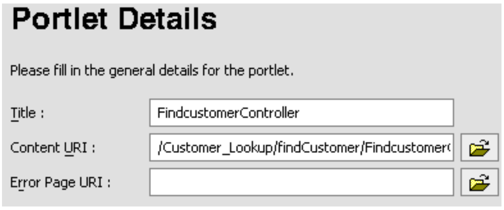
State : ☐ Minimizable ☐ Maximizable ☐ Deletable

Available Modes : ☐ Help ☐ Edit

Note that Title and Content URL are already filled in (Figure 4-28) based upon the page flow name.

3. Change the Title to Find Customer Information.

Figure 4-28 Portlet Details Detail



Portlet Details

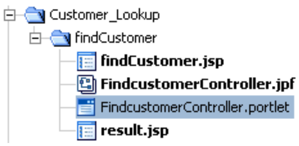
Please fill in the general details for the portlet.

Title :	FindcustomerController
Content URI :	/Customer_Lookup/findCustomer/Findcustomer(
Error Page URI :	

4. Click Finish.

The new portlet appears in the Customer_Lookup/findCustomer folder (Figure 4-29).

Figure 4-29 FindCustomerController Portlet on File Tree



Attach the Backing File

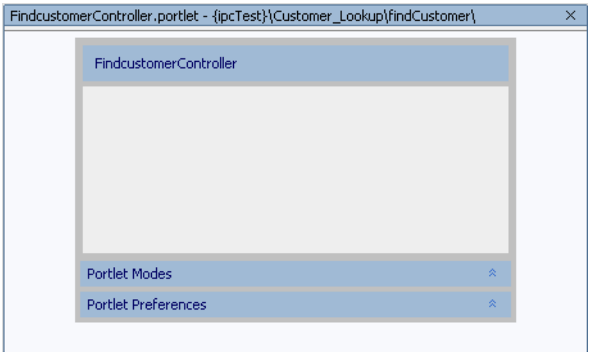
Now, you will attach the backing file `receiveCustomerInfo.java` to the portlet (`FindCustomerController.portlet`). As described in Step 2: Create the Backing Files, this file contains business logic that will fetch a payload containing information regarding the customer whose name was submitted in the Customer Lookup portlet.

To attach the backing file, use this procedure:

1. In the file tree, double-click `FindCustomerController.portlet`.

   The portlet appears in the Portlet Designer (Figure 4-30).

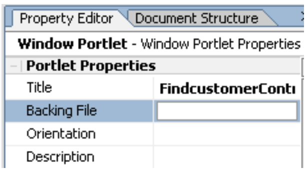
Figure 4-30 Portlet in Portlet Designer


FindcustomerController.portlet - {ipcTest}\Customer_Lookup\findCustomer\
FindcustomerController
Portlet Modes
Portlet Preferences

2. In the Portlet Window section of the Property Editor, select Backing File.

   The field opens to accept text entry (Figure 4-31).

Figure 4-31 Property Editor; Backing File Selected



3. In the field, enter `backing.receiveCustomerInfo` and press [Tab].

Note: *Do not* include the filetype (`.java`). WebLogic Workshop interprets the dot-separators in this field as filepath separators and, if `.java` is included, it will look for a file called `java` in a non-existent folder called `backing/FindCustomerBacking`.

The backing file is attached.

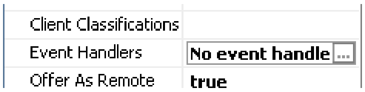
Create the Event Handlers

Finally, you need to create the event handlers for this portlet. In this step, you will select the event handler types, identify the portlets to which they will listen, and define the actions they will fire when they detect the prescribed event.

To create the event handlers, use this procedure:

1. With the portlet open in the Portlet Designer (Figure 4-30), locate in the Portlet Window section of the Property Editor the property Event Handlers.
2. Mouse over the Event Handlers data entry field (it will read "No event handlers") to open the field for edit (Figure 4-32).

Figure 4-32 Event Handler Property Open for Edit

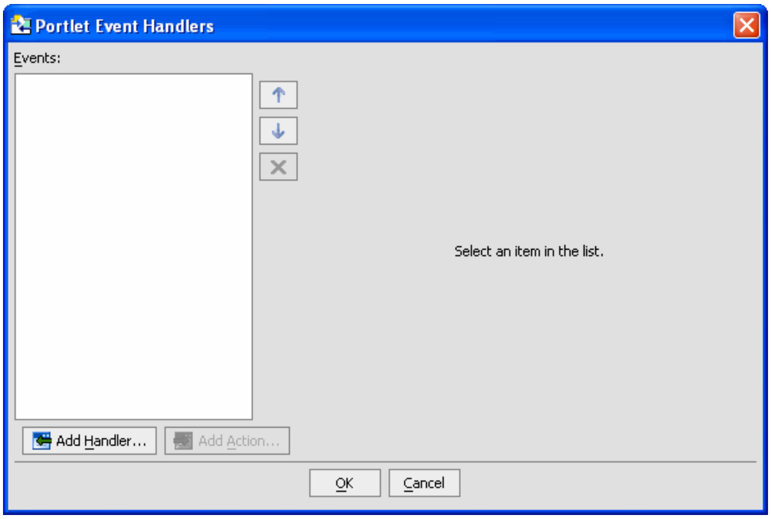


Client Classifications	
Event Handlers	No event handle ...
Offer As Remote	true

3. Click the ellipsis (...) next to the data entry field.

   The Event Handler tool appears (Figure 5.).

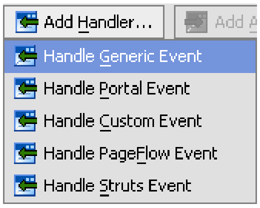
Figure 4-33 Event Handler Tool



4. Click Add Handler...

   A menu opens, listing the available event handlers (Figure 4-34).

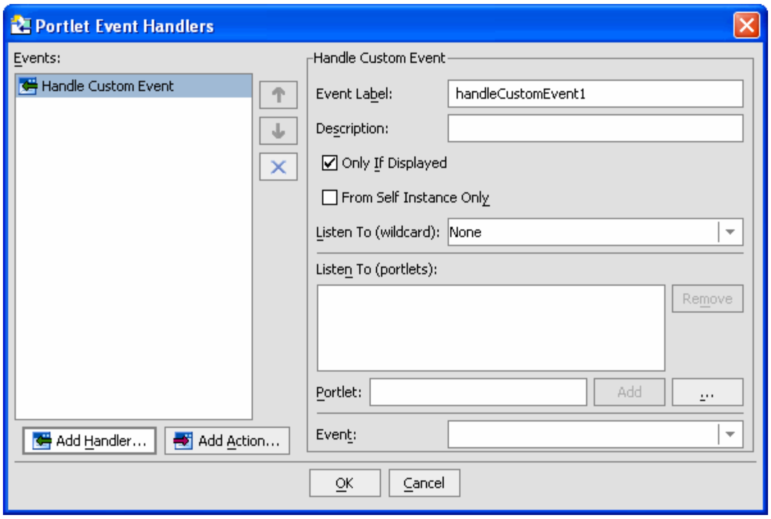
Figure 4-34 Event Handler Menu



5. Select Handle Custom Event.

   Handle Custom Event appears in the Events list and additional fields appear on the dialog box (Figure 4-35).

Figure 4-35 Event Handler Tool with Handle Custom Event Selected



Handle Custom Event

Event Label: handleCustomEvent1

Description:

☑ Only If Displayed

☐ From Self Instance Only

Listen To (wildcard): None

Listen To (portlets):

Portlet:

Event:

6. Fill out the Handle Custom Event fields as described in Table 4-1.

Table 4-1 Handle Custom Event Fields

In...	Enter or Select...
Event Label	handleCustomEvent1 (this default value should appear when you select Handle Custom Event from the Add Handler list)
Only if Displayed	*Uncheck*

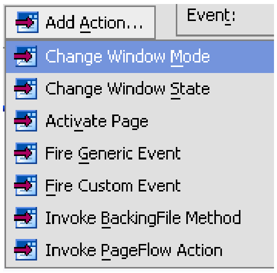
Table 4-1 Handle Custom Event Fields

Listen To (wildcard):	Any
Event:	myCustomEvent

7. Click Add Action...

   A list of actions appears (Figure 4-36).

Figure 4-36 Add Action Menu

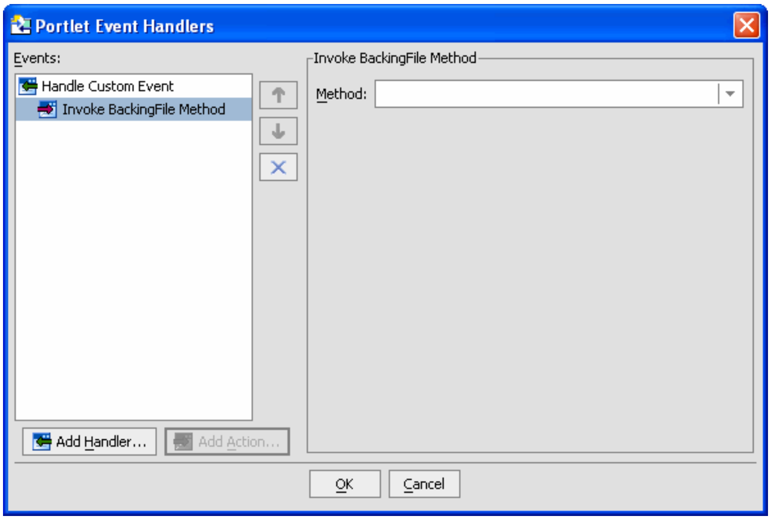


8. Select Invoke BackingFile Method

Note: This option appears only if a backing file is attached to the portlet; if it does not appear, ensure that you completed Attach the Backing File successfully.

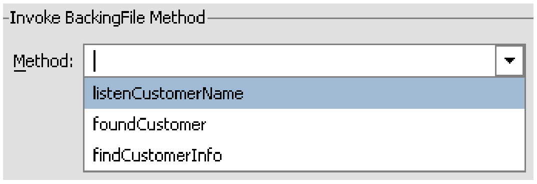
   Invoke BackingFile Method appears in the Event Handler list, and the right pane of the dialog box reformats as shown in Figure 4-37.

Figure 4-37 Event Handler with Invoke BackingFile Method Added as Action



9. Click the Method drop-down control to display a list of available methods (Figure 4-38).

Figure 4-38 Invoke BackingFile Method Available Method List



10. Select listenCustomerName.

11. Repeat steps 1 through 7.

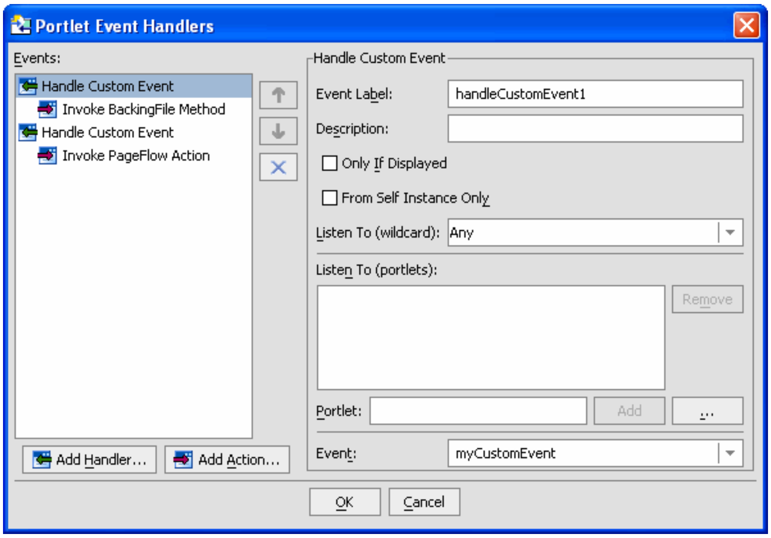
12. Click Add Action to open the Action drop-down menu and select Invoke PageFlow Method.

    The dialog box refreshes and the Action edit box appears in the right-hand pane.

13. Click the drop-down control and select findCustomerInfo.

    The Event Handler tool should look like the example in Figure 4-39.

Figure 4-39 Event Handler Tool Completed for findCustomer Portlet


Portlet Event Handlers

Events:
- Handle Custom Event
  - Invoke BackingFile Method
- Handle Custom Event
  - Invoke PageFlow Action

Handle Custom Event

Event Label: handleCustomEvent1

Description:

☐ Only If Displayed

☐ From Self Instance Only

Listen To (wildcard): Any

Listen To (portlets):

Portlet:

Event: myCustomEvent

14. Click OK to close the dialog box and press [Tab] to close the Event Handlers data field.

15. Click the Save Files icon or open the File menu and select Save All.

Step 4: Create the Orders Lookup Portlet

The Orders Lookup Portlet surfaces information about orders pertaining to the customer submitted in the Customer Lookup portlet. This portlet is constructed identically to the Customer Lookup portlet, with some different data necessary for it to execute properly.

Note: To save you time and to reduce the chance of entry errors when creating these files, we have provided the necessary code in this document. If you are reading this online (HTML), you can simply cut and paste the code into the appropriate empty file. If you are using a hardcopy or PDF version of this document and cannot obtain an online copy, you can type the code *exactly as it appears* in the code samples.

Create the Page Flow (FindOrderController.jpf)

The FindOrder page flow (`FindOrderController.jpf`) establish the order in which pages are rendered in the Order Lookup portlet. This page flow executes after a name is submitted on the Customer Lookup Portlet. You create the Find Order page flow (`FindOrderController.jpf`), exactly as you create the Find Customer page flow, described in Create the Page Flow (FindCustomerController.jpf), although obviously with different data included.

To create the FindOrderController.jpf, use this procedure:

1. Right-click `Orders_LookUp` and select New > Page Flow.

   The Page Flow Wizard - Page Flow Name dialog box appears.

2. In Page Flow Name: enter findOrder.

   Note that Controller Name: is automatically filled-in with the text FindOrderController.

3. Click Next

   The Page Flow Wizard - Select Page Flow Type dialog box appears.

4. Select Basic page flow and click Create.

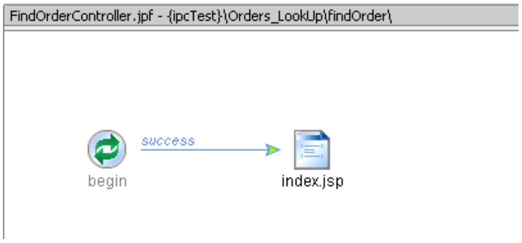
   The following happens:

   – A folder called `findOrder` appears under `Orders_LookUp`.

   – Two files, `FindOrderController.jpf` and `index.jsp` appear under `Orders_LookUp/findOrder`.

   – A schematic representation of the default page flow appears in the Page Flow Designer (Figure 4-40).

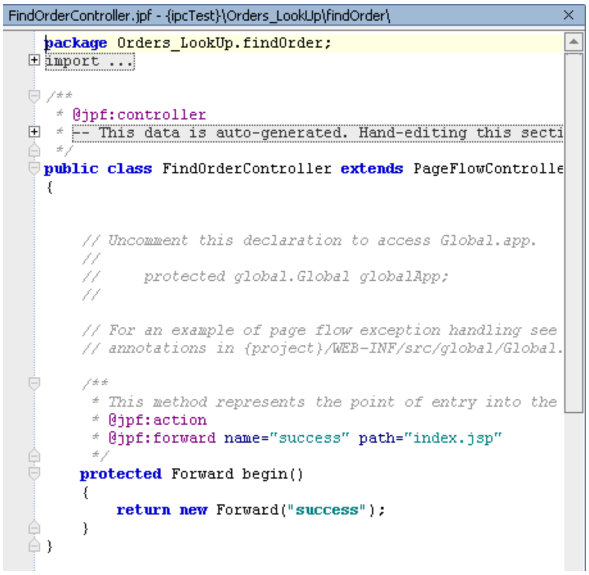
Figure 4-40 Default Page Flow Schematic in Page Flow Designer — Flow View


FindOrderController.jpf - {ipcTest}\Orders_LookUp\findOrder\
success
begin
index.jsp

At the bottom of the Page Flow Designer, click Source View

The Page Flow Designer Source View appears, revealing the code for the default page flow (Figure 4-41).

Figure 4-41 Default Page Flow Code in Source View



```
FindOrderController.jpf - {ipcTest}\Orders_LookUp\findOrder\
package Orders_LookUp.findOrder;
import ...

/**
 * @jpf:controller
 * -- This data is auto-generated. Hand-editing this secti
 */
public class FindOrderController extends PageFlowControlle
{

    // Uncomment this declaration to access Global.app.
    //
    //     protected global.Global globalApp;
    //

    // For an example of page flow exception handling see
    // annotations in {project}/WEB-INF/src/global/Global.

    /**
     * This method represents the point of entry into the
     * @jpf:action
     * @jpf:forward name="success" path="index.jsp"
     */
    protected Forward begin()
    {
        return new Forward("success");
    }
}
```

Select the entire code listing and delete it.

Copy the code sample in Listing 4-7 and paste it into the Page Flow Designer Source View.

Listing 4-7 FindOrderController.jpf Code Sample

```
package Orders_LookUp.findOrder;
import com.bea.wlw.netui.pageflow.FormData;
import com.bea.wlw.netui.pageflow.Forward;
import com.bea.wlw.netui.pageflow.PageFlowController;

/**
 * @jpf:controller
 * @jpf:view-properties view-properties::
 * <!-- This data is auto-generated. Hand-editing this section is not
 * recommended. -->
 * <view-properties>
 * <pageflow-object id="pageflow:/Orders_LookUp/findOrder/
 *    FindOrderController.jpf"/>
 * <pageflow-object id="action:begin.do">
 *   <property value="80" name="x"/>
 *   <property value="100" name="y"/>
 * </pageflow-object>
 * <pageflow-object id="action:findOrders.do#Orders_LookUp.findOrder.
 *    FindOrderController.FindOrdersForm">
 *   <property value="420" name="x"/>
 *   <property value="100" name="y"/>
 * </pageflow-object>
 * <pageflow-object id="action-call:@page:findOrderForm.jsp@#@action:
 *    findOrders.do#Orders_LookUp.findOrder.FindOrderController.
 *    FindOrdersForm@">
 *   <property value="276,330,330,384" name="elbowsX"/>
 *   <property value="92,92,92,92" name="elbowsY"/>
 *   <property value="East_1" name="fromPort"/>
 *   <property value="West_1" name="toPort"/>
 * </pageflow-object>
 * <pageflow-object id="page:findOrderForm.jsp">
 *   <property value="240" name="x"/>
 *   <property value="100" name="y"/>
 * </pageflow-object>
 * <pageflow-object id="page:orderResults.jsp">
 *   <property value="600" name="x"/>
 *   <property value="100" name="y"/>
 * </pageflow-object>
 * <pageflow-object id="forward:path#success#findOrderForm.jsp#@action:
 *    begin.do@">
 *   <property value="116,160,160,204" name="elbowsX"/>
 *   <property value="92,92,92,92" name="elbowsY"/>
 *   <property value="East_1" name="fromPort"/>
 *   <property value="West_1" name="toPort"/>
 *   <property value="success" name="label"/>
 * </pageflow-object>

 * <pageflow-object id="forward:path#success#orderResults.jsp#@action:
 *    findOrders.do#Orders_LookUp.findOrder.FindOrderController.
 *    FindOrdersForm@">
 *   <property value="456,510,510,564" name="elbowsX"/>
 *   <property value="92,92,92,92" name="elbowsY"/>
 *   <property value="East_1" name="fromPort"/>
 *   <property value="West_1" name="toPort"/>
 *   <property value="success" name="label"/>
 * </pageflow-object>
 * <pageflow-object id="formbeanprop:Orders_LookUp.findOrder.
 *    FindOrderController.FindOrdersForm#customer_name#java.lang.String"/>
 * <pageflow-object id="formbeanprop:Orders_LookUp.findOrder.
 *    FindOrderController.FindOrdersForm#order_id#java.lang.String"/>
 * <pageflow-object id="formbean:Orders_LookUp.findOrder.FindOrderController.
 *    FindOrdersForm"/>
 * <pageflow-object id="action-call:@page:orderResults.jsp@#@action:findOrders.
 *    do#Orders_LookUp.findOrder.FindOrderController.FindOrdersForm@">
 *   <property value="564,510,510,456" name="elbowsX"/>
 *   <property value="92,92,81,81" name="elbowsY"/>
```

```
 *   <property value="West_1" name="fromPort"/>
 *   <property value="East_0" name="toPort"/>
 * </pageflow-object>
 * </view-properties>
 * ::
 */
public class FindOrderController extends PageFlowController
{
    // Uncomment this declaration to access Global.app.
    //
    //     protected global.Global globalApp;
    //

    // For an example of page flow exception handling see the example
    // "catch" and "exception-handler"
    // annotations in {project}/WEB-INF/src/global/Global.app

    /**
     * This method represents the point of entry into the pageflow
     * @jpf:action
     * @jpf:forward name="success" path="findOrderForm.jsp"
     */
    protected Forward begin()
    {
        return new Forward("success");
    }


    /**
     * @jpf:action
     * @jpf:forward name="success" path="orderResults.jsp"
     */
    protected Forward findOrders(FindOrdersForm form)
    {
        return new Forward("success");
    }

    /**
     * FormData get and set methods may be overwritten by the Form Bean editor.
     */
    public static class FindOrdersForm extends FormData
    {
        private String order_id;

        private String customer_name;

        public void setCustomer_name(String customer_name)
        {
            this.customer_name = customer_name;
        }

        public String getCustomer_name()
        {
            return this.customer_name;
        }

        public void setOrder_id(String order_id)
        {
            this.order_id = order_id;
        }

        public String getOrder_id()
        {
            return this.order_id;
        }
    }
}
```

5. Save the file either by clicking the Save icon or opening the File menu and selecting Save.

Note: You might see two "File could not be found" warnings (indicated by a green squiggle) in the Page Flow Designer. You can ignore them, as they refer to files that you will create in the next two steps.

`FindOrderController.jpf` is created.

Create findOrderForm.jsp

Next, you need to create one of the two JSPs required for the page flow to run successfully. `findOrder.jsp` provides the presentation logic for the initial portlet view. From this portlet, you can launch the page flow by submitting a customer name. In this example, the page flow is launched when the Order LookUp portlet hears a name submitted on the Customer Lookup portlet.

To create f`indOrderForm.jsp`, use this procedure.

1. Right-click findOrder in the file tree and select New>JSP.

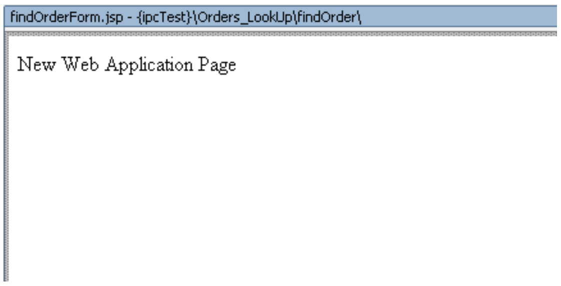
   The New File dialog box appears. Ensure that Web User Interface and JSP File are selected.

2. In File Name:, enter findOrderForm and click Create.

   The following happens:

- The file `findOrderForm.jsp` appears in the file tree under `Orders_LookUp/findOrders`.
- The text "New Web Application Page" appears in the JSP Designer (Figure 4-42). This is the default JSP.

Figure 4-42 JSP Designer with Default JSP Design View


findOrderForm.jsp - {ipcTest}\Orders_LookUp\findOrder\

New Web Application Page

3. At the bottom of the JSP Designer, select Source View.

   The JSP Designer Source View appears, revealing the code for the default JSP.

4. Select the *entire* default code listing and delete it.
5. Copy the code sample in Listing 4-8 and paste it into the JSP Designer.

Listing 4-8 `findOrderForm.jsp` Sample Code

```
<%@ page language="java" contentType="text/html;charset=UTF-8"%>

<%@ taglib uri="netui-tags-databinding.tld" prefix="netui-data"%>
<%@ taglib uri="netui-tags-html.tld" prefix="netui"%>
<%@ taglib uri="netui-tags-template.tld" prefix="netui-template"%>

<netui:html>
    <head>
        <title>
            Find Order
        </title>
    </head>
    <body>
        <p> Find Order </p>

       <%= //session.getAttribute("customerName")
       %>
```

```
        <netui:form action="findOrders">
            <table>
                <tr valign="top">
                    <td>Customer_name:</td>
                    <td>
                    <netui:textBox dataSource="{actionForm.customer_name}"/>
                    </td>
                </tr>
                <tr valign="top">
                    <td>Order_id:</td>
                    <td>
                    <netui:textBox dataSource="{actionForm.order_id}"/>
                    </td>
                </tr>
            </table>
            <br/>&nbsp;
            <netui:button value="Find Orders" type="submit"/>
        </netui:form>
    </body>
</netui:html>
```

---

6. Save the file either by clicking the Save icon or opening the File menu and selecting Save.

   `findOrderForm.jsp` is created.

Create orderResults.jsp

The second JSP you need to create is `orderResults.jsp`. This file provides the presentation logic for displaying the results of the order lookup for the customer specified in the Customer Name or Order ID fields; however, in this example, it will return order information for the customer name submitted in the Customer Lookup portlet.

To create orderResults.jsp, use this procedure.

1. Under `Orders_LookUp/findCustomer`, double-click `index.jsp`.

   The text “New Web Application Page” appears in the JSP Designer. This is the default JSP.

2. At the bottom of the JSP Designer, click Source View.

   The JSP Designer refreshes to show the default JSP code.

3. Select the entire default JSP code listing and delete it.

4. Copy the code sample in Listing 4-9 and paste it into the JSP Designer.

Listing 4-9 orderResults.jsp Code Sample

```
<%@ page language="java" contentType="text/html;charset=UTF-8"%>
<%@ taglib uri="netui-tags-databinding.tld" prefix="netui-data"%>
<%@ taglib uri="netui-tags-html.tld" prefix="netui"%>
<%@ taglib uri="netui-tags-template.tld" prefix="netui-template"%>
<netui:html>
    <head>
        <title>
            Web Application Page
        </title>
    </head>
    <body>
        <p>

       Customer Name: <%=session.getAttribute("customerName")
       %>
       <%
        session.setAttribute("customerName", "");
       %>
        <netui:form action="findOrders">

            <netui:label value="{actionForm.customer_name}"/>
        </netui:form>
        <br>
        Order 001<br>
        Order 002<br>
        Order 003<br>
        Order 004<br>
        <br>

    </body>
</netui:html>
```

5. Save the file either by clicking the Save or opening the File menu and selecting Save.
6. Right-click index.jsp in the file tree to open a context menu and select Rename.

   The node is opened for renaming.
7. Rename the file `orderResults`.

Note: If you reopen `FindCustomerController.jpf` after saving both JSPs, you will see that the warnings described in the note following Listing 4-7 no longer appear.

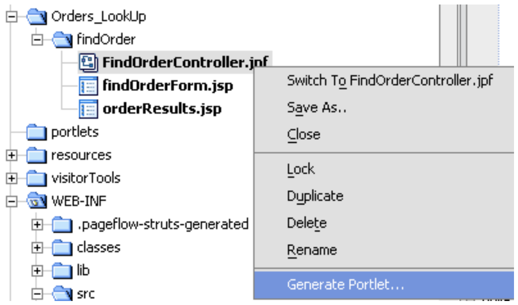
Create the Portlet File

The portlet file, `FindOrderController.portlet`, is the container for the page flow and the JSPs you just created. This is the file to which you'll attach the backing file and implement the event handlers necessary to achieve interportlet communications.

To create the Order Lookup portlet, use this procedure.

1. Right-click `FindOrderController.jpf` to open a context menu and select Generate portlet... (Figure 4-43)

Figure 4-43 Generating `FindOrderController.portlet`



   The Portlet Wizard - Portlet Details dialog box appears. Note that the Title and Portlet URI fields are already completed.

2. Change the Title to Find Orders.
3. Click Finish.

   The portlet is created and appears in the file tree under `Orders_LookUp/findOrder`.

Attach the Backing File

Now, you will attach the backing file `ListenCustomerName.java` to the portlet (`FindOrderController.portlet`). As described in Step 2: Create the Backing Files, this file contains business logic that will fetch information regarding the orders posted by the customer whose name was submitted in the Customer Lookup portlet.

To attach the backing file, use this procedure:

1. In the file tree, double-click `FindCustomerController.portlet`.

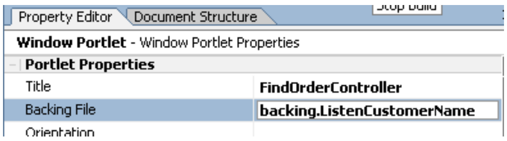
   The portlet appears in the Portlet Designer.

2. In the Portlet Window section of the Property Editor, select Backing File.

   The field opens to accept text entry.

3. In the field, enter `backing.ListenCustomerName` and press [Tab].

Figure 4-44 Attaching `ListenCustomerName.java` to the Portlet



Window Portlet - Window Portlet Properties
Portlet Properties

Title	FindOrderController
Backing File	backing.ListenCustomerName
Orientation	

Note: *Do not* include the filetype (`.java`). WebLogic Workshop interprets the dot-separators in this field as filepath separators and, if `.java` is included, will look for a file called `java` in a non-existent folder called `backing/FindCustomerBacking`.

4. Save the file by either clicking the Save icon or by opening the File Menu and selecting Save.

   The backing file is attached.

Create the Event Handlers

Finally, you need to create the event handlers for this portlet. In this step, you will select the event handler types, identity the portlets to which they will listen, and define the actions they will fire upon detecting the prescribed event.

To create the event handlers, use this procedure:

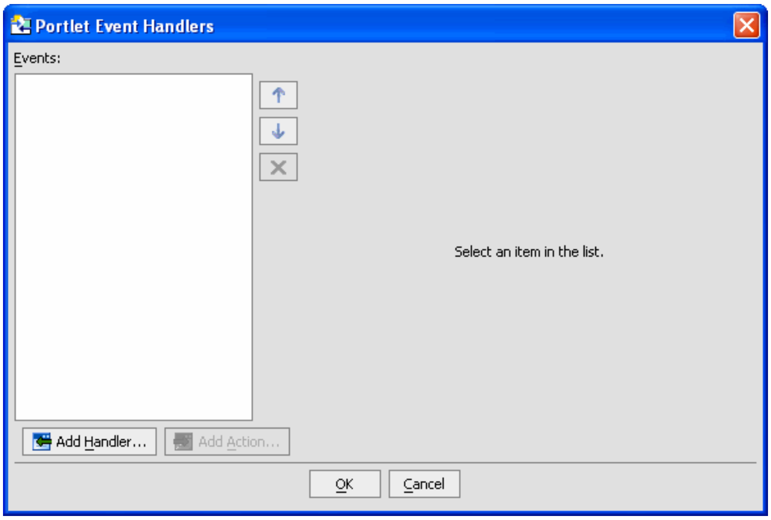
1. With the portlet open in the Portlet Designer (Figure 4-30), locate in the Portlet Window section of the Property Editor the property Event Handlers.

2. Mouse over the Event Handlers' data entry field (it will read "No event handlers") to open the field for edit.

3. Click the ellipsis (...) next to the data entry field.

   The Event Handler tool appears (Figure 4-45).

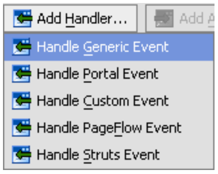
Figure 4-45 Event Handler Tool



4. Click Add Handler...

   A menu opens, listing the available event handlers (Figure 4-46).

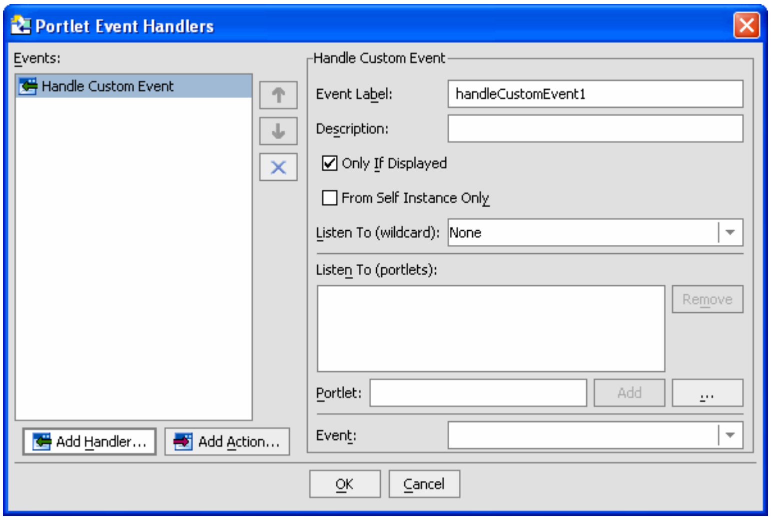
Figure 4-46 Event Handler Menu



5. Select Handle Custom Event.

   Handle Custom Event appears in the Events list and additional fields appear on the dialog box (Figure 4-47).

Figure 4-47 Event Handler Tool with Handle Custom Event Selected



Event Label: handleCustomEvent1
Description:
[✓] Only If Displayed
[ ] From Self Instance Only
Listen To (wildcard): None
Listen To (portlets):
Portlet:
Event:

6. Fill out the Handle Custom Event fields as described in Table 4-2.

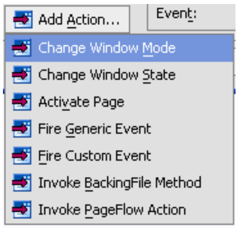
Table 4-2 Handle Custom Event Fields

In...	Enter or Select...
Event Label	handleCustomEvent1 (this default value should appear when you select Handle Custom Event from the Add Handler list)
Only if Displayed	*Uncheck*
Listen To (wildcard):	Any
Event:	myCustomEvent

7. Click Add Action...

   A list of actions appears (Figure 4-48).

Figure 4-48 Add Action Menu

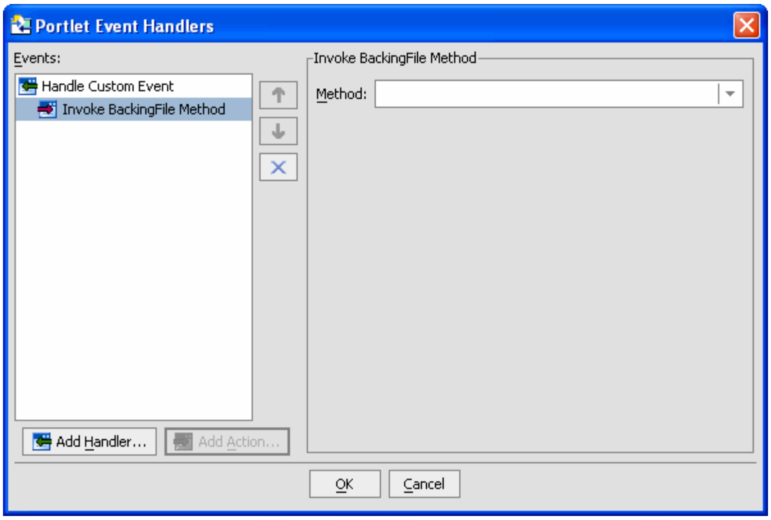


8. Select Invoke BackingFile Method.

Note: This option only appears if a backing file is attached to the portlet; if it does not appear, ensure that you completed Attach the Backing File successfully).

Invoke BackingFile Method appears in the Event Handler list and the right pane of the dialog box reformats as shown in Figure 4-49.

Figure 4-49 Event Handler with Invoke BackingFile Method Added as Action



9. Click the Method drop-down control to display a list of available methods.

10. Select listenCustomerName.

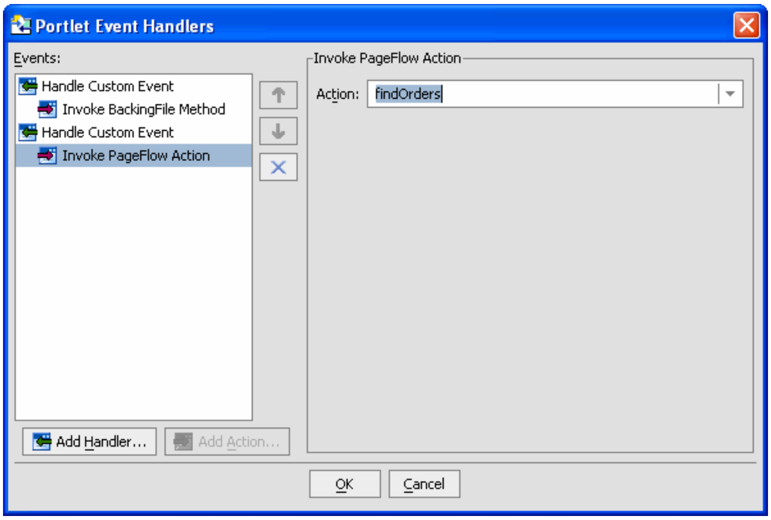
11. Repeat steps 1 through 8; however, at step 8. select Invoke PageFlow Action.

12. Click the Method drop-down control to display a list of available methods.

13. Select findOrders.

    The Event Handler tool should look like the example in Figure 4-50.

Figure 4-50 Event Handler Tool Completed for findCustomer Portlet



Action: findOrders

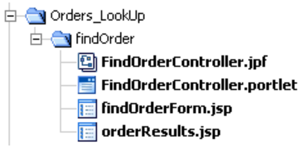
14. Click OK.

15. Click the Save Files icon or open the File menu and select Save All.

Summary

At this point, you have completed the Order Lookup Portlet for the ipcText Web application. To verify that you've created all files necessary, compare the `Orders_LookUp` folder in your file tree with that in Figure 4-51. Your file tree should match the tree in the example.

Figure 4-51 `Orders_LookUp` Folder with Order Lookup Portlet Complete



```
Orders_LookUp
  findOrder
    FindOrderController.jpf
    FindOrderController.portlet
    findOrderForm.jsp
    orderResults.jsp
```

Step 5: Create the Customer Lookup Portlet

The Customer Lookup portlet is the last portlet you will create for this exercise. This portlet is where a user will enter a customer's name and submit it to the system. This submission will create events that the portlets created in Step 3: Create the Customer Lookup Results Portlet and Step 4: Create the Orders Lookup Portlet will hear and fire appropriate actions.

Note: To save you time and to reduce the chance of entry errors when creating these files, we have provided the necessary code in this document. If you are reading this online (HTML), you can simply cut and paste the code into the appropriate empty file. If you are using a hardcopy or PDF version of this document and cannot obtain an online copy, you can type the code *exactly as it appears* in the code samples.

Create the Page Flow (FindCustomerController.jpf)

`FindCustomerController.jpf` establishes the order in which pages are rendered in the Customer Lookup portlet. This page flow executes after a name is submitted on this portlet. You create the Customer Lookup page flow exactly as you create the other page flows in this exercise, although obviously with different data included.

Note: Do not confuse this `FindCustomerController.jpf` with `FindCustomerController.jpf` created in Step 3: Create the Customer Lookup Results Portlet. The identical names are an idiosyncrasy of this demonstration.

To create the FindCustomerController.jpf, use this procedure:

1. Right-click `portlets` and select New>Page Flow.

   The Page Flow Wizard - Page Flow Name dialog box appears.

2. In Page Flow Name: enter findCustomer.

   Note that Controller Name: is automatically filled-in with the text FindCustomerController.

3. Click Next

   The Page Flow Wizard - Select Page Flow Type dialog box appears.

4. Select Basic page flow and click Create.

   The following happens:

   – A folder called `findCustomer` appears under `portlets` in the file tree.

   – Two files, `FindCustomerController.jpf` and `index.jsp` appear under `portlets/findCustomer`.

   – A default page flow schematic appears in the Page Flow Designer.

5. At the bottom of the Page Flow Designer, select Source View.

   The view in the Page Flow Designer refreshes to reveal the code for the default page flow.

6. Highlight the *entire* contents of the Page Flow Designer and delete it, so that the Source View is empty.

7. Copy the code sample in Listing 4-10 and paste it into the empty Page Flow Designer Source View.

Listing 4-10 FindCustomerController.jpf Code Sample (Second Version)

```
package portlets.findCustomer;
import com.bea.wlw.netui.pageflow.FormData;
import com.bea.wlw.netui.pageflow.Forward;
import com.bea.wlw.netui.pageflow.PageFlowController;

/**
 * @jpf:controller
 * @jpf:view-properties view-properties::
 * <!-- This data is auto-generated. Hand-editing this section is not
 * recommended. -->
 * <view-properties>
 * <pageflow-object id="pageflow:/portlets/findCustomer/
 * FindCustomerController.jpf"/>
 * <pageflow-object id="action:begin.do">
 *   <property value="80" name="x"/>
 *   <property value="100" name="y"/>
 * </pageflow-object>
 * <pageflow-object id="page:index.jsp">
 *   <property value="240" name="x"/>
 *   <property value="100" name="y"/>
 * </pageflow-object>
 * <pageflow-object id="forward:path#success#index.jsp#@action:begin.do@">
 *   <property value="116,160,160,204" name="elbowsX"/>
 *   <property value="92,92,92,92" name="elbowsY"/>
 *   <property value="East_1" name="fromPort"/>
 *   <property value="West_1" name="toPort"/>
 *   <property value="success" name="label"/>
 * </pageflow-object>
 * <pageflow-object id="page:result.jsp">
 *   <property value="580" name="x"/>
 *   <property value="100" name="y"/>
 * </pageflow-object>
 * <pageflow-object id="action:findCustomer.do#portlets.findCustomer.
 * FindCustomerController.FindCustomerForm">
 *   <property value="420" name="x"/>
 *   <property value="100" name="y"/>
 * </pageflow-object>
 * <pageflow-object id="forward:path#success#result.jsp#@action:findCustomer.
 *    do#portlets.findCustomer.FindCustomerController.FindCustomerForm@">
 *   <property value="456,500,500,544" name="elbowsX"/>
 *   <property value="92,92,92,92" name="elbowsY"/>
 *   <property value="East_1" name="fromPort"/>
 *   <property value="West_1" name="toPort"/>
 *   <property value="success" name="label"/>
 * </pageflow-object>
 * <pageflow-object id="formbean:portlets.findCustomer.FindCustomerController.
 *    FindCustomerForm"/>
 * <pageflow-object id="formbeanprop:portlets.findCustomer.
 *    FindCustomerController.FindCustomerForm#name#java.lang.String"/>
 * <pageflow-object id="formbeanprop:portlets.findCustomer.
 *    FindCustomerController.FindCustomerForm#company#java.lang.String"/>
 * <pageflow-object id="action-call:@page:index.jsp@#@action:findCustomer.
 *    do#portlets.findCustomer.FindCustomerController.FindCustomerForm@">
 *   <property value="276,330,330,384" name="elbowsX"/>
 *   <property value="92,92,92,92" name="elbowsY"/>
 *   <property value="East_1" name="fromPort"/>
 *   <property value="West_1" name="toPort"/>
 * </pageflow-object>
 * </view-properties>
 * ::
 */
public class FindCustomerController extends PageFlowController
{

    // Uncomment this declaration to access Global.app.
    //
    //     protected global.Global globalApp;
    //


    // For an example of page flow exception handling see the example
    // "catch" and "exception-handler"
    // annotations in {project}/WEB-INF/src/global/Global.app

    /**
     * This method represents the point of entry into the pageflow
     * @jpf:action
     * @jpf:forward name="success" path="index.jsp"
     */
    protected Forward begin()
    {
        return new Forward("success");
    }

    /**
     * @jpf:action
     * @jpf:forward name="success" path="result.jsp"
     */
    protected Forward findCustomer(FindCustomerForm form)
    {
        return new Forward("success");
    }

    /**
     * FormData get and set methods may be overwritten by the Form Bean editor.
     */
    public static class FindCustomerForm extends FormData
    {
        private String company;

        private String name;

        public void setName(String name)
        {
            this.name = name;
        }

        public String getName()
        {
            return this.name;
        }

        public void setCompany(String company)
        {
            this.company = company;
        }

        public String getCompany()
        {
            return this.company;
        }
    }
}
```

Note: You might see a “File could not be found” warning (indicated by a green squiggle) in the Page Flow Designer. You can ignore this warning, as it refers to a file, `result.jsp`, that you will create in Create result.jsp.

8. Save the file either by clicking the Save icon or by opening the File menu and selecting Save.

Update index.jsp

Since this is the "launching pad" for the three page flows in the Web application (that is, when a name is submitted in this portlet, through interportlet communications, page flow actions will occur on the portlets created in Step 3: Create the Customer Lookup Results Portlet and Step 4: Create the Orders Lookup Portlet, as well as this one), this exercise will conform to the best practice of beginning this page flow with `index.jsp` file.

This JSP provides the presentation logic for the Customer Lookup portlet before a customer name is submitted. Once a name is submitted, that logic will be superseded by the presentation logic in `result.jsp`, which we will create in the next step. When you created the page flow in Create the Page Flow (FindCustomerController.jpf), the template for `index.jsp` was also created and placed in `portlets/findCustomer`.

To create `index.jsp`, use this procedure.

1. Double-click `index.jsp` in the file tree under portlets/findCustomer.

   The file opens, in Design View, in the JSP Designer.

2. At the bottom of the JSP Designer, click Source View.

   The JSP Designer refreshes to reveal the default `index.jsp` code.

3. In the JSP Designer, select the entire code listing and delete it.

   Copy the code sample in Listing 4-11 and paste it into the JSP Designer.

Listing 4-11 index.jsp Code Sample

```
<%@ page language="java" contentType="text/html;charset=UTF-8"%>
<%@ taglib uri="netui-tags-databinding.tld" prefix="netui-data"%>
<%@ taglib uri="netui-tags-html.tld" prefix="netui"%>
<%@ taglib uri="netui-tags-template.tld" prefix="netui-template"%>
<netui:html>
    <head>
        <title>
            Web Application Page
        </title>
    </head>
    <body>
        <p> New Web Application Page </p>
        <netui:form action="findCustomer">
            <table>
                <tr valign="top">
                    <td>Company:</td>
                    <td>
                    <netui:textBox dataSource="{actionForm.company}"/>
                    </td>
                </tr>
                <tr valign="top">
                    <td>Name:</td>
                    <td>
                    <netui:textBox dataSource="{actionForm.name}"/>
                    </td>
                </tr>
            </table>
            <br/>&nbsp;
            <netui:button value="findCustomer" type="submit"/>
        </netui:form>
    </body>
</netui:html>
```

4. Save the file by either clicking the Save icon or opening the File menu and selecting Save.

Create result.jsp

Next, create the `result.jsp` file. This file contains the presentation logic that will format the display of information returned to this portlet upon successful rendering of the FindCustomerController page flow. This logic will supersede that contained in `index.jsp`.

To create `result.jsp`, use this procedure.

1. Right-click `portlets/findCustomer` and select New>JSP File.

   The New File dialog box appears. Ensure that Web User Interface and JSP File are selected. If not, select them now.

2. In File name:, enter result and click Create.

   The following happens:

   – `result.jsp` appears under `portlets/findCustomer`.

   – A default JSP (characterized by the text “New Web Application Page” appears in the JSP Designer.

3. At the bottom of the JSP Designer, click Source View.

The JSP Designer refreshes, revealing the code for the default JSP.

4. Select the entire default JSP code listing and delete it.
5. Copy the code sample in Listing 4-12 and paste it into the JSP Designer.

Listing 4-12 result.jsp Code Sample

```
<%@ page language="java" contentType="text/html;charset=UTF-8"%>
<%@ taglib uri="netui-tags-databinding.tld" prefix="netui-data"%>
<%@ taglib uri="netui-tags-html.tld" prefix="netui"%>
<%@ taglib uri="netui-tags-template.tld" prefix="netui-template"%>
<netui:html>
    <head>
        <title>
            Find Customer Result
        </title>
    </head>
    <body>
        <br>
            Customer Information:
        <br>
        <br>

        <netui:form action="findCustomer">

        <b>Name: <netui:label value="{actionForm.name}"/>
        <br>
        Address: 4001 Discovery Drive<br>
        City: Boulder<br>
        State: CO<br></b>
        </b>
        </netui:form>
    </body>
</netui:html>
```

6. Save the file by either clicking the Save icon or by opening the File menu and selecting Save.

Note: If you reopen `FindCustomerController.jpf` after saving `result.jsp`, you will see that the warning described in the note following Listing 4-10 is gone.

Create the Portlet File

The portlet file, `FindOrderController.portlet`, is the container for the page flow and the JSPs you just created. This is the file to which you'll attach the backing file and implement the event handlers necessary to achieve interportlet communications.

To create the Customer Lookup portlet, use this procedure.

1. Under `portlets/findCustomer`, right-click `FindCustomerController.jpf` to open the file's context menu.
2. Select Generate Portlet

   The Portlet Details dialog box appears. Note that Title and Content URL are already filled-in, based upon the page flow name.
3. Change the Title to Find Customer.
4. Click Finish.

   The new portlet appears in the `portlets/findCustomer` folder.

Attach the Backing File

Now, you will attach the backing file `FindCustomerBacking.java` to the portlet (`FindCustomerController.portlet`).

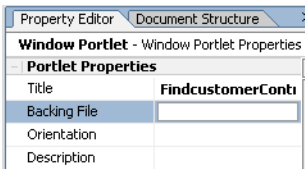
To attach the backing file, use this procedure:

1. In the file tree, double-click `FindCustomerController.portlet`.

   The portlet appears in the Portlet Designer.
2. In the Portlet Window section of the Property Editor, select Backing File.

   The field opens to accept text entry.

Figure 4-52 Property Editor; Backing File Selected



Title	FindcustomerConti
Backing File	
Orientation	
Description	

3. In the field, enter `backing.FindCustomerBacking` and press [Tab].

Note: *Do not* include the filetype (`.java`). WebLogic Workshop interprets the dot-separators in this field as filepath separators and, if `.java` is included, will look for a file called `java` in a non-existent folder called `backing/FindCustomerBacking` folder.

The backing file is attached.

Create the Event Handler

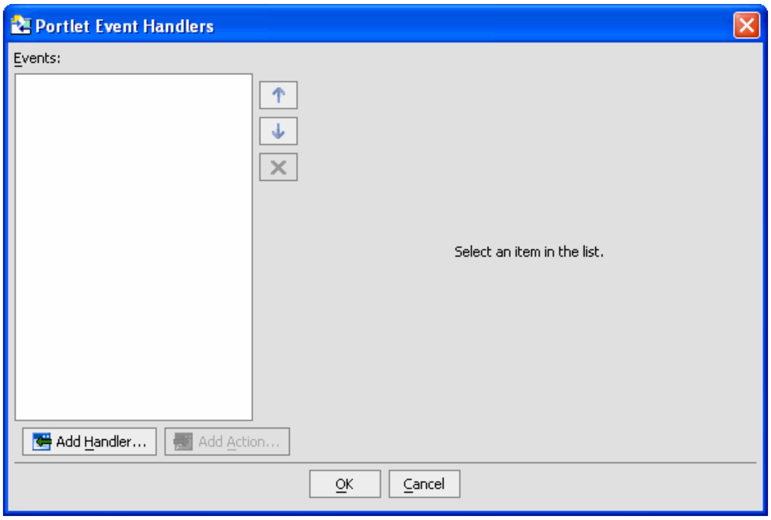
Finally, you need to create the event handler for this portlet (unlike the portlets created in Step 3: Create the Customer Lookup Results Portlet and Step 4: Create the Orders Lookup Portlet, this portlet uses only one event handler). In this step, you will select the event handler type, identity the portlets to which it will listen, and define the action it will fire upon detecting the prescribed event.

To create the event handler, use this procedure:

1. With the portlet open in the Portlet Designer, locate in the Portlet Window section of the Property Editor the property Event Handlers.
2. Mouse over the Event Handlers' data entry field (it will read "No event handlers") to open the field for edit.
3. Click the ellipsis (...) next to the data entry field.

   The Event Handler tool appears (Figure 4-45).

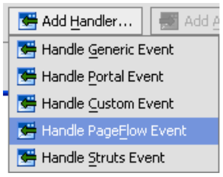
Figure 4-53 Event Handler Tool



4. Click Add Handler...

   A menu opens, listing the available event handlers (Figure 4-54).

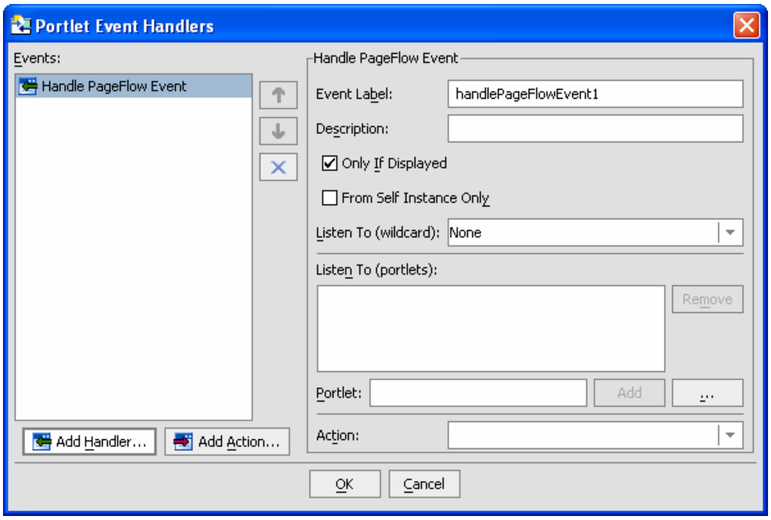
Figure 4-54 Event Handler Menu



5. Select Handle PageFlow Event.

   Handle PageFlow Event appears in the Events list and additional fields appear on the dialog box (Figure 4-55).

Figure 4-55 Event Handler Tool with Handle PageFlow Event Selected



Event Label: handlePageFlowEvent1
Description:
☑ Only If Displayed
☐ From Self Instance Only
Listen To (wildcard): None
Listen To (portlets):
Portlet:
Action:

6. Fill out the Handle Custom Event fields as described in Table 4-3.

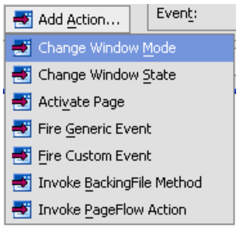
Table 4-3 Handle Custom Event Fields

In...	Enter or Select...
Event Label	handlePageFlowEvent1 (this default value should appear when you select Handle Custom Event from the Add Handler list)
Only if Displayed	*Uncheck*
Listen To (wildcard):	None
Action:	findCustomer

7. Click Add Action...

   A list of actions appears (Figure 4-56).

Figure 4-56 Add Action Menu

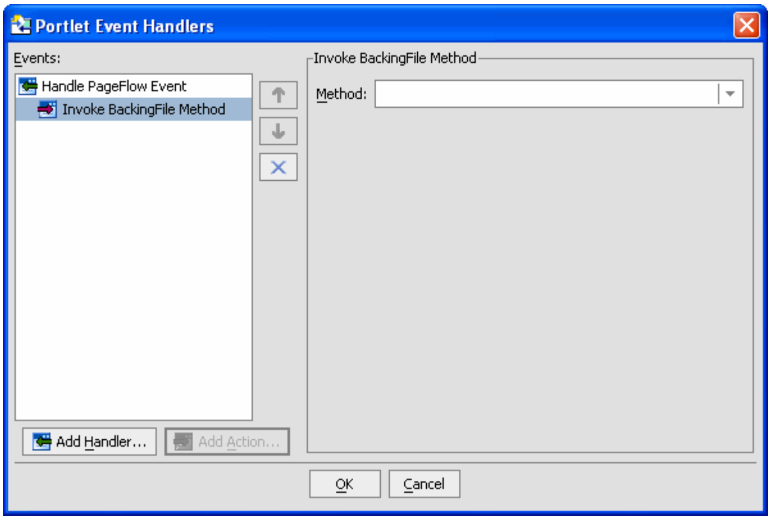


8. Select Invoke BackingFile Method.

Note: This option only appears if a backing file is attached to the portlet; if it does not appear, ensure that you completed Attach the Backing File successfully).

Invoke BackingFile Method appears in the Event Handler list and the right pane of the dialog box reformats as shown in Figure 4-57.

Figure 4-57 Event Handler with Invoke BackingFile Method Added as Action

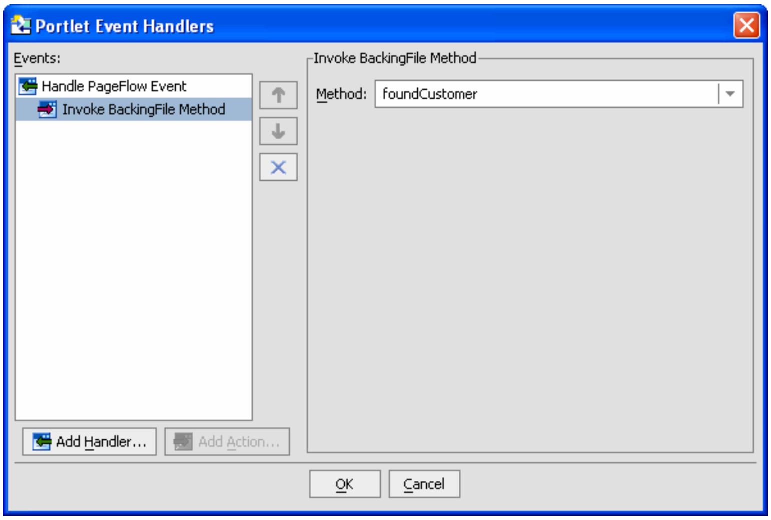


9. Click the Method drop-down control to display a list of available methods.

10. Select foundCustomer.

The Event Handler tool should look like the example in Figure 4-58.

Figure 4-58 Event Handler Tool Completed for findCustomer Portlet



Invoke BackingFile Method

Method: foundCustomer

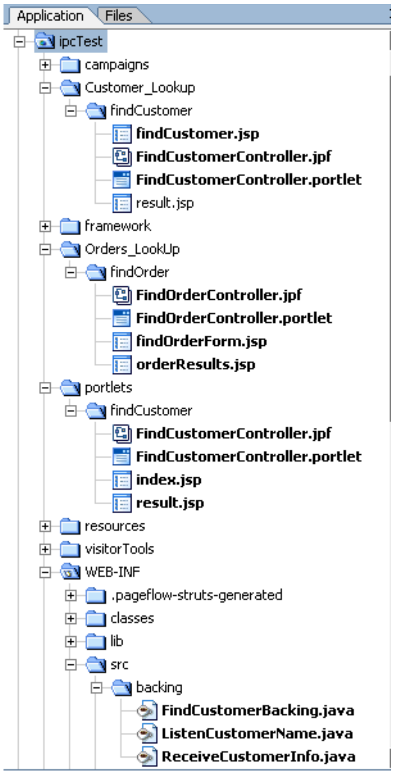
11. Click OK.

12. Click the Save Files icon or open the File menu and select Save All.

Summary

With the event handlers created for the Customer Lookup portlet, you have finished creating all three portlets for this exercise. To verify that you have created all of the necessary files, please compare the file tree in your current version of WebLogic Workshop to the example in Figure .

Figure 4-59 File Tree After All Portlets are Complete



```
Application  Files
ipcTest
    campaigns
    Customer_Lookup
        findCustomer
            findCustomer.jsp
            FindCustomerController.jpf
            FindCustomerController.portlet
            result.jsp
    framework
    Orders_LookUp
        findOrder
            FindOrderController.jpf
            FindOrderController.portlet
            findOrderForm.jsp
            orderResults.jsp
    portlets
        findCustomer
            FindCustomerController.jpf
            FindCustomerController.portlet
            index.jsp
            result.jsp
    resources
    visitorTools
    WEB-INF
        .pageflow-struts-generated
        classes
        lib
        src
            backing
                FindCustomerBacking.java
                ListenCustomerName.java
                ReceiveCustomerInfo.java
```

The next step is to add the three portlets you just created to a portal, as described in Step 6: Create and Populate the Portal File.

Step 6: Create and Populate the Portal File

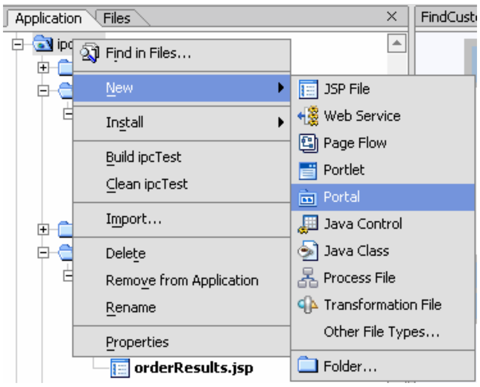
In this step, you will create the portal, characterized by a `.portal` file, that will contain the conversant portlets and add these portlets to the portal.

Create the Portal File

To create the portal file, use this procedure.

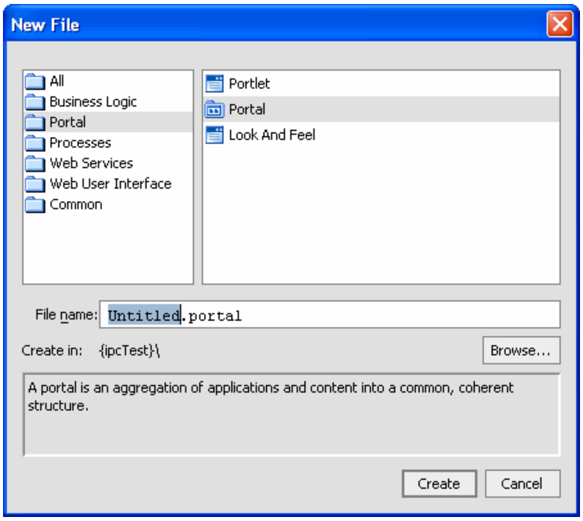
1. In the file tree, right-click the Web application name (ipcTest) and select New>Portal (Figure 4-60).

Figure 4-60 Creating a New Portal



The New File dialog box appears (Figure 4-61).

Figure 4-61 New File Dialog Box Configured for Creating a New Portal



File name: Untitled.portal

Create in: {ipcTest}\

A portal is an aggregation of applications and content into a common, coherent structure.

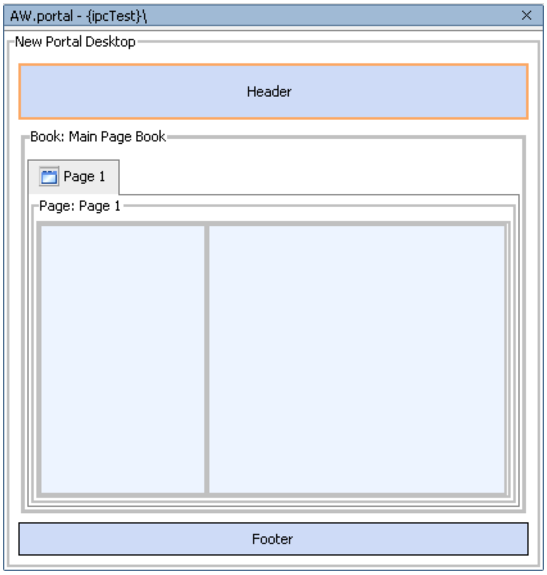
2. In File name, enter AW.

3. Click Create.

   The file `AW.portal` is created and the following appears:

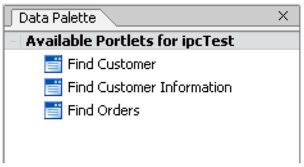
   – A portal schematic will appear in the Portal Designer (Figure 4-62).

Figure 4-62 Portal Schematic Appears in the Portal Designer



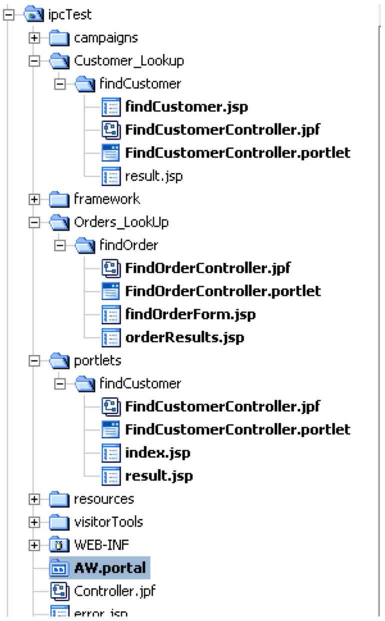
– The three portlets created for this exercise appear in the Data Palette (Figure 4-63).

Figure 4-63 Data Palette Showing New Portlets



– The file `AW.portal` appears in the file tree at the same level as the individual portlet folders (Figure 4-64).

Figure 4-64 AW.portal in the File Tree



```
ipcTest
    campaigns
    Customer_Lookup
        findCustomer
            findCustomer.jsp
            FindCustomerController.jpf
            FindCustomerController.portlet
            result.jsp
    framework
    Orders_LookUp
        findOrder
            FindOrderController.jpf
            FindOrderController.portlet
            findOrderForm.jsp
            orderResults.jsp
    portlets
        findCustomer
            FindCustomerController.jpf
            FindCustomerController.portlet
            index.jsp
            result.jsp
    resources
    visitorTools
    WEB-INF
    AW.portal
    Controller.jpf
    error.jsp
```

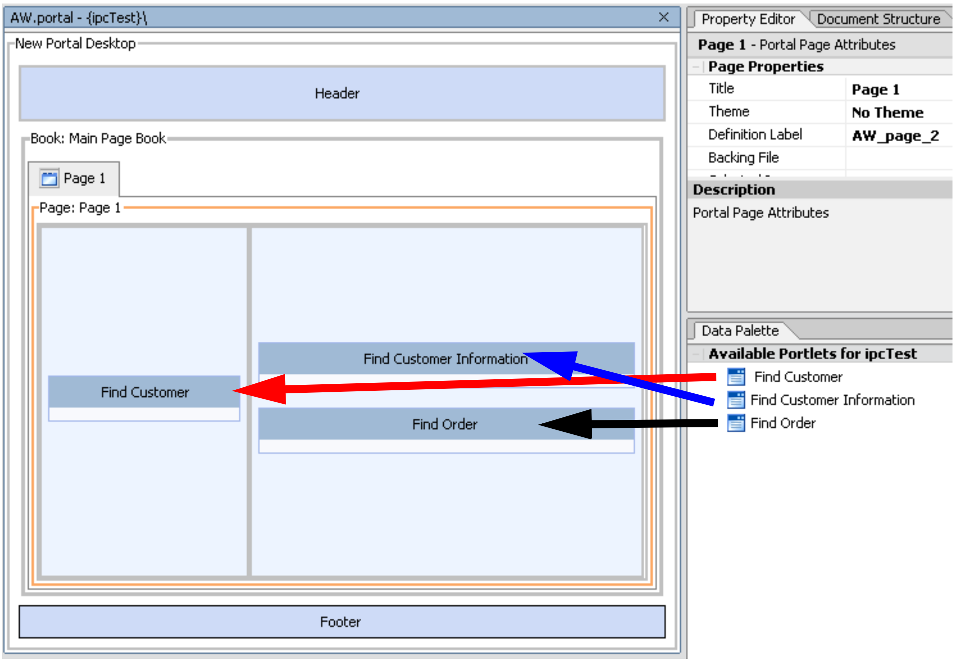
4. Save the file by either clicking the Save icon or by opening the file menu and selecting Save.

Add Portlets to the Portal

The last step in actually creating the portal application is to add the newly created portlets to the newly created portal. To do so, use this procedure.

1. Select each portlet in the Data Palette and drag it to the appropriate position in the portal, as illustrated in Figure 4-65.

Figure 4-65 Adding Portlets to the Portal



Page 1 - Portal Page Attributes

Page Properties

Title	**Page 1**
Theme	**No Theme**
Definition Label	**AW_page_2**
Backing File	

Description

Portal Page Attributes

2. Save the portal either by clicking the Save icon or by opening the File menu and selecting Save.

Step 7: Test the Portal

This final step will determine if you have achieved interportlet communications as intended by this exercise. You will start WebLogic Server, run the portal, and submit a user's name. When you submit the name, the portlets should communicate between themselves and return data particular to that user.

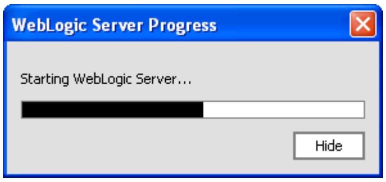
To test the portal, use this procedure.

Note: Ensure that `AW.portal` is open in WebLogic Workshop.

1. Start the server by opening the Tools menu and selecting WebLogic Server>Start WebLogic Server.

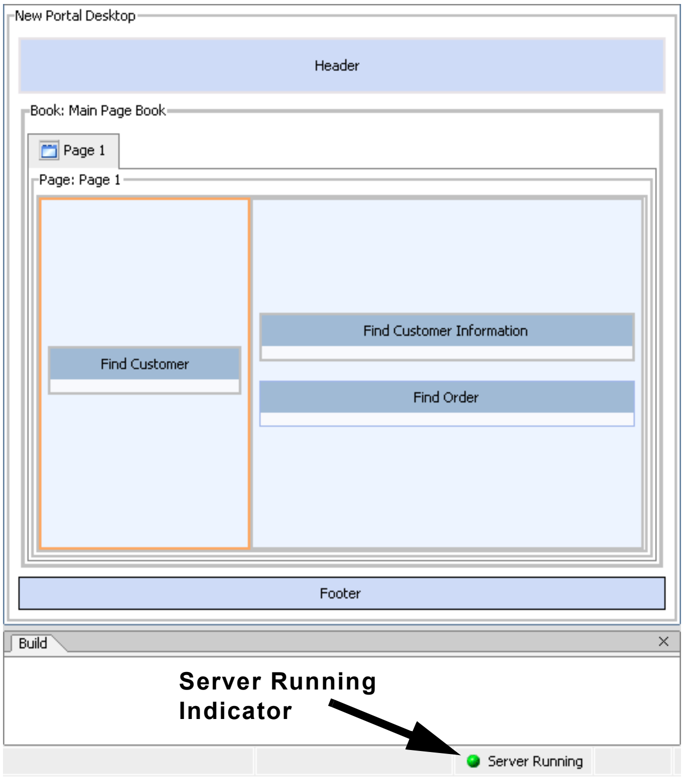
   A progress meter (Figure 4-66), indicating that the server is starting, appears.

Figure 4-66 Server Startup Progress Meter



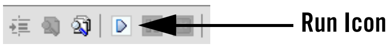
The server takes a few moments to start. When the progress meter closes, the server is running. Note the green Server Running indicator at the bottom of WebLogic Workshop (Figure 4-67).

Figure 4-67 Server Running Indicator



2. Run the Portal by clicking the Run icon on the Toolbar (Figure 4-68).

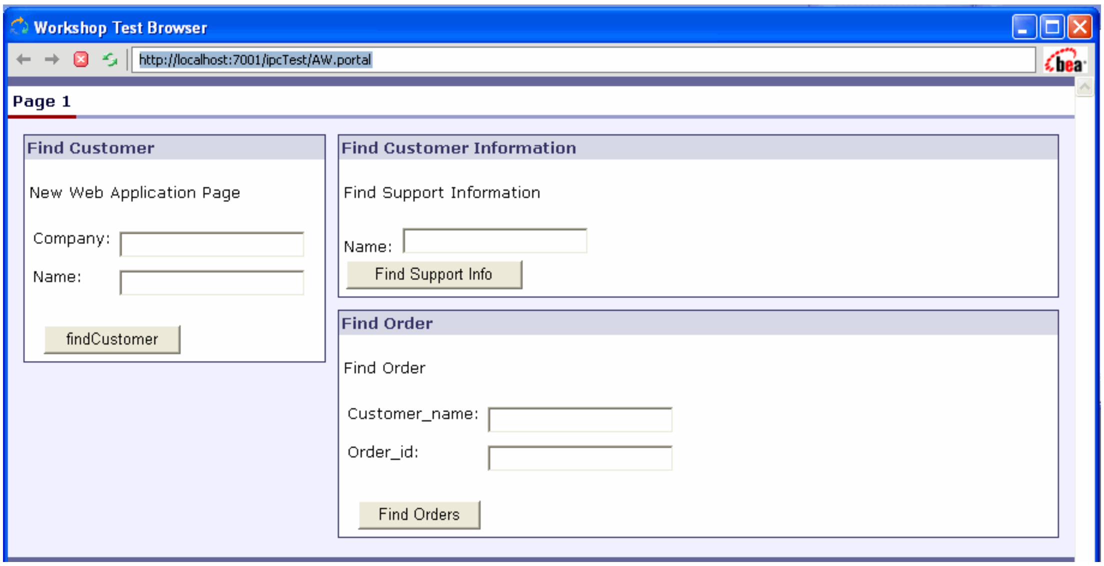
Figure 4-68 Run Icon



Note: Clicking this icon causes the debugger to run before the portal renders.

The portal opens in a test browser (Figure 4-69).

Figure 4-69 Portal Opens in a Test Browser



Find Customer

New Web Application Page

Company:

Name:

findCustomer

Find Customer Information

Find Support Information

Name:

Find Support Info

Find Order

Find Order

Customer_name:

Order_id:

Find Orders

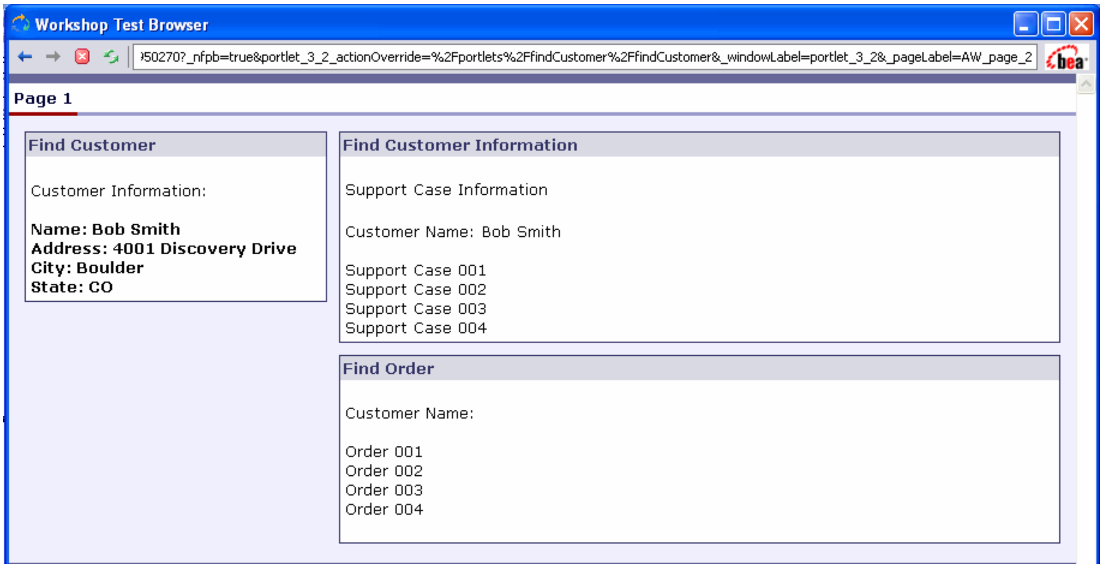
Test the Portal

To test the portal (that is, to demonstrate the IPC has been successfully achieved), do the following:

1. In the Find Customer Portlet, enter XYZ Systems in Company and Bob Smith in Name.
2. Click Submit.

   All three portlets refresh with Bob Smith's customer and order information (Figure 4-70).

Figure 4-70 Test Browser with Customer and Order Information


Workshop Test Browser
)50270?_nfpb=true&portlet_3_2_actionOverride=%2Fportlets%2FfindCustomer%2FfindCustomer&_windowLabel=portlet_3_2&_pageLabel=AW_page_2
Page 1

Find Customer

Customer Information:

Name: Bob Smith
Address: 4001 Discovery Drive
City: Boulder
State: CO

Find Customer Information

Support Case Information

Customer Name: Bob Smith

Support Case 001
Support Case 002
Support Case 003
Support Case 004

Find Order

Customer Name:

Order 001
Order 002
Order 003
Order 004

CHAPTER 5

Understanding Backing Files

Backing files allow you to programatically add functionality to a portlet by implementing (or extending) a Java class, which enables preprocessing (for example, authentication) prior to rendering the portal controls. Backing files can be attached to portals either by using WebLogic Workshop or coding them directly into a `.portlet` file.

This section is primer on backing files. It includes information on the following subjects:

- What are Backing Files?
- Which Controls Support Backing Files?
- How Backing Files are Executed
- Thread Safety with Backing Files
- Creating a Backing File
- Adding a Backing File to a Portlet

What are Backing Files?

Backing files are simple Java classes that implement the `com.bea.netuix.servlets.controls.content.backing.JspBacking` interface or extend the `com.bea.netuix.servlets.controls.content.backing.AbstractJspBacking` `interface` abstract class. The methods on the interface mimic the controls lifecycle methods (see "How Backing Files are Executed") and are invoked at the same time the controls lifecycle methods are invoked.

Which Controls Support Backing Files?

At this time, the following controls support backing files:

- Desktops
- Books
- Pages
- Portlets

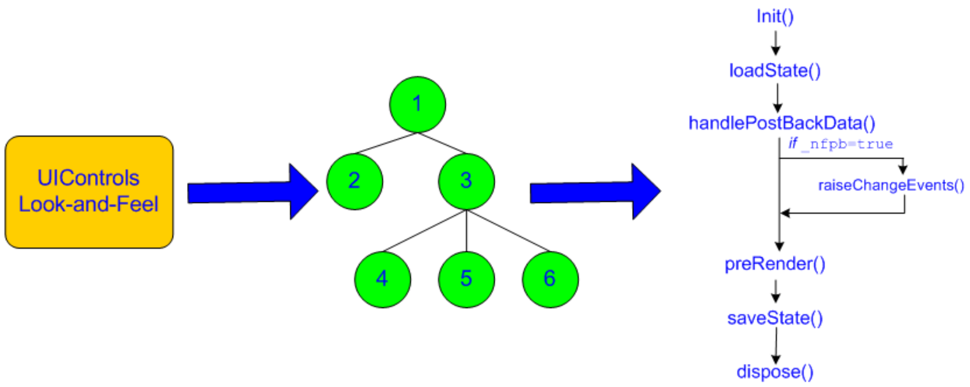
How Backing Files are Executed

All backing files are executed before and after the JSP is called. In its lifecycle, each backing file calls these methods:

- `init()`
- `handlePostBackData()`
  - `raiseChangeEvents()`
- `preRender()`
- `dispose()`

Figure 5-1 illustrates the lifecycle of a backing file.

Figure 5-1 Backing File Lifecycle



On every request, the following occurs:

1. All `init()` methods are called on all backing files on an "in order" basis (that is, in the order they appear in the tree). This method gets called whether or not the control (that is, portal, page, book, or desktop) is on an active page.

2. Next, if the operation is a postback *and* the control (a portlet, page, or book) is on a visible page, all `handlePostbackData()` methods are called. In other words if portlet is on a page but its parent page is not active, then this method will not get called.

   – If `_nfpb="true"` is set in the request parameter of any `handlePostbackData()` methods called, `raiseChangeEvents()` is called. This method causes events to fire.

3. Next, all `preRender()` methods are called for all controls on an active (visible) page.

4. Next, the JSPs get called and are rendered on the active page by the `<render:beginRender>` JSP tag. Rendering is stopped with the `<render:endRender>` tag.

5. Finally, the `dispose()` method gets called on the backing file.

Note: `roadstead()` and `savviest()`, shown in Figure 5-1 are part of the control lifecyle, not the backing file lifecycle.

Other Execution Notes

If the backing file is part of a floated portlet, when that portlet is floated, only *its* contents are executed.

If a book is embedded within a portlet, then the book would get called; however, if the book is the parent of the portlet then it would not get called as it is not contained within the portlet.

Thread Safety with Backing Files

A new instance of a backing file is created per request, so you don't have to worry about thread safety issues. New Java VMs are specially tuned for short-lived objects, so this is not the performance issues it once was in the past. Also, `JspContent` controls support a special type of backing file that allows you to specify whether or not the backing file is thread safe. If this value is set to `true`, only one instance of the backing file is created and shared across all requests.

Creating a Backing File

As previously discussed, a backing file must be an implementation of `com.bea.netuix.servlets.controls.content.backing.JspBacking` interface or an extension of the com.bea.netuix.servlets.controls.content.backing.AbstractJspBacking abstract class. You only need to modify these files as necessary to implement the backing functionality you desire.

Listing 5-1 is the backing file used in Establishing IPC by Using Custom and Page Flow Events In this example, the AbstractJspBacking class is extended to provide the backing functionality required by the portlet.

Listing 5-1 Backing File Example

```
package backing;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import com.bea.netuix.events.Event;
import com.bea.netuix.events.CustomEvent;
import com.bea.netuix.servlets.controls.content.backing.AbstractJspBacking;


public class ListenCustomerName extends AbstractJspBacking
{
    public void listenCustomerName( HttpServletRequest request,
       HttpServletResponse response, Event event)
    {

        CustomEvent customEvent = (CustomEvent) event;

        String message = (String) customEvent.getPayload();

        HttpSession mySession = request.getSession();

        mySession.setAttribute("customerName", message);

    }

}
```

You should follow these guidelines when creating a backing file:

- Ensure `netuix_servlet.jar` is included in the in the project classpath, otherwise compilation errors will occur.

- When implementing the `init()` method, avoid any heavy processing.

Adding a Backing File to a Portlet

As a best practice, you should always store backing files in the web application's `WEB-INF/src/backing` directory, as the /src directory is the first place the application looks for a backing file (if you are storing the first backing file for a web application, you will need to create the `/backing` directory under `WEB-INF/src`).

Adding the Backing File by Using WebLogic Workshop

You can add a backing file to a portlet either from within WebLogic Workshop or by coding it directly into the file to which you are attaching it. For all other portlet types, simply specify the backing file in the **Backing File** field under the **General Properties** section of the Property Editor, as shown in Figure 5-2. You need to specify the backing directory and, following a dot-separator, *just* the backing file name. Do not include the backing file extension; for example enter this:

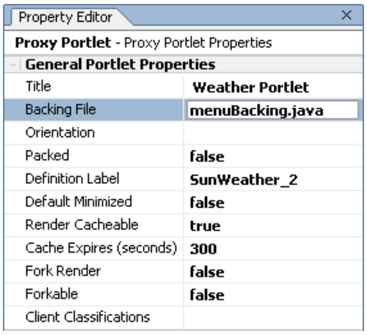
```
backing.ListenCustomerName
```

Not this:

```
backing.ListenCustomerName.java
```

If you include the file extension, the application will interpret is as the file name—because the file path is specified by a dot-separator—and look for a non-existent file called `java` in a non-existent directory called `ListenCustomerName` (or whatever your backing file is named).

Figure 5-2 Adding a Backing File by Using the IDE



Proxy Portlet - Proxy Portlet Properties

General Portlet Properties

Title	Weather Portlet
Backing File	menuBacking.java
Orientation	
Packed	false
Definition Label	SunWeather_2
Default Minimized	false
Render Cacheable	true
Cache Expires (seconds)	300
Fork Render	false
Forkable	false
Client Classifications	

Adding the Backing File Directly to the .portlet Code

To add the backing file by coding it into a `.portlet` file, use the `backingFile` parameter within the `<netuix:jspContent>` element, as shown in Listing 5-2.

Listing 5-2 Adding a Backing File to a .portlet File

```
<netuix:content>
   <netuix:jspContent
   backingFile="portletToPortlet.pageFlowSelectionDisplayOnly.menu.
      backing.MenuBacking"
   contentUri="/portletToPortlet/pageFlowSelectionDisplayOnly/menu/
      menu.jsp"/>
</netuix:content>
```