



BEA WebLogic Workshop™

Portal Tutorials

Copyright

Copyright © 2004-2005 BEA Systems, Inc. All Rights Reserved.

Restricted Rights Legend

This software and documentation is subject to and made available only pursuant to the terms of the BEA Systems License Agreement and may be used or copied only in accordance with the terms of that agreement. It is against the law to copy the software except as specifically allowed in the agreement. This document may not, in whole or in part, be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine readable form without prior consent, in writing, from BEA Systems, Inc.

Use, duplication or disclosure by the U.S. Government is subject to restrictions set forth in the BEA Systems License Agreement and in subparagraph (c)(1) of the Commercial Computer Software-Restricted Rights Clause at FAR 52.227-19; subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013, subparagraph (d) of the Commercial Computer Software--Licensing clause at NASA FAR supplement 16-52.227-86; or their equivalent.

Information in this document is subject to change without notice and does not represent a commitment on the part of BEA Systems. THE SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. FURTHER, BEA Systems DOES NOT WARRANT, GUARANTEE, OR MAKE ANY REPRESENTATIONS REGARDING THE USE, OR THE RESULTS OF THE USE, OF THE SOFTWARE OR WRITTEN MATERIAL IN TERMS OF CORRECTNESS, ACCURACY, RELIABILITY, OR OTHERWISE.

Trademarks or Service Marks

BEA, BEA WebLogic Server, Jolt, Tuxedo, and WebLogic are registered trademarks of BEA Systems, Inc. BEA Builder, BEA Campaign Manager for WebLogic, BEA eLink, BEA Liquid Data for WebLogic, BEA Manager, BEA WebLogic Commerce Server, BEA WebLogic Enterprise, BEA WebLogic Enterprise Platform, BEA WebLogic Enterprise Security, BEA WebLogic Express, BEA WebLogic Integration, BEA WebLogic JRockit, BEA WebLogic Personalization Server, BEA WebLogic Platform, BEA WebLogic Portal, BEA WebLogic Server Process Edition, BEA WebLogic Workshop and How Business Becomes E-Business are trademarks of BEA Systems, Inc.

All other trademarks are the property of their respective companies.

Contents

About This Document

- What You Need to Knowvii
- Product Documentation on the dev2dev Web Site.vii
- Related Informationviii
- Contact Us!viii
- Documentation Conventionsix

Portal Tutorials

- Tutorial: Building Your First Portal.1
 - Tutorial Goals1
 - Tutorial Overview1
 - Steps in This Tutorial1
 - Step 1: Start WebLogic Workshop, Open an Application, and Start WebLogic Server .2
 - Start WebLogic Workshop2
 - Open the Sample Portal Application2
 - Start WebLogic Server.3
 - Step 2: Create a Portal.3
 - Create a Portal File3
 - Add a Page to the Portal4
 - Add a Portlet to the Portal4
 - View the Portal4
 - What You Can Do Next.5

Tutorial: Changing a Portal's Look & Feel and Navigation	5
Tutorial Goals.	5
Tutorial Overview	6
Steps in This Tutorial	6
Step 1: Start WebLogic Workshop, Open an Application, and Start WebLogic Server. 6	
Start WebLogic Workshop	7
Open the Sample Portal Application	7
Start WebLogic Server	7
Step 2: Change the Portal Look & Feel and Navigation	8
Open the Sample Portal	8
View the Portal in a Browser	8
Add a Book to the Main Page Book	8
Change the Portal's Look and Feel	9
Change the Portal's Navigation.	9
View the Modified Portal in a Browser	9
Tutorial: Showing Personalized Content in a Portlet	9
Tutorial Goals.	10
Tutorial Overview	10
Steps in This Tutorial	10
Step 1: Start WebLogic Workshop, Open an Application, and Start WebLogic Server 11	
Start WebLogic Workshop	11
Open the Sample Portal Application	11
Start WebLogic Server	12
Step 2: Create a User Profile Property Set.	12
Create a Property Set File	12
Add a Property to the Property Set	12
Step 3: Create Two Users.	13
Start the WebLogic Administration Portal	13

Create Two Users and Set Their Profile Preferences	13
Step 4: Load Content	14
Step 5: Create Two Content Selectors	15
Create a "modern" content selector	15
Create a "classic" content selector	16
Step 6: Create a Portlet	17
Create a Portlet	17
Step 7: Modify the JSP	18
Step 8: Test the Personalized Portlet	20
View the Sample Portal	21
Log in as One User and View the Portlet	21
Log in as the Other User and View the Portlet	21
Tutorial: Creating a Login Control Page Flow Using the Wizard	21
Tutorial Overview	21
Steps in This Tutorial	22
Step 1: Start WebLogic Workshop, Open an Application, and Start WebLogic Server	22
Start WebLogic Workshop	22
Open the Sample Portal Application	22
Start WebLogic Server	23
Step 2: Create and Configure a Page Flow	23
Step 3: Create a Portlet and Edit the Login Form	29
Tutorial: Using Page Flows Inside Portlets	37
Tutorial Goals	37
Tutorial Overview	38
Steps in This Tutorial	38
Step 1: Start WebLogic Workshop, Open an Application, and Start WebLogic Server	39
Start WebLogic Workshop	39
Open the Sample Portal Application	39

Start WebLogic Server	39
Step 2: Add a Portal to the Application	40
Add a Portal	40
Step 3: Create a Simple Navigation Page Flow	40
Create a Page Flow called simpleFlow.	40
Test the Page Flow in a Portlet	47
JSP Examples	47
index.jsp.	48
page2.jsp	48
Step 4: Create Portlets that Communicate	48
Create two Page Flows	49
Edit Page Flows	50
Edit j1 Page Flow	50
Edit j2 Page Flow	51
Create Additional JSPs	53
Create listening.jsp	53
Create simpleForm.jsp	53
Create and Place Portlets in the Portal	55
Set Properties on Portlets	56
Step 5: Create an EJB Control Page Flow Portlet	58
Create a New Page Flow.	59
Place the Page Flow into a Portlet, Place Portlet in a Portal	60

About This Document

The following tutorials highlight the process and features of developing a portal with the WebLogic Workshop Portal Extensions:

- [Tutorial: Building Your First Portal](#)
- [Tutorial: Changing a Portal's Look & Feel and Navigation](#)
- [Tutorial: Showing Personalized Content in a Portlet](#)
- [Tutorial: Creating a Login Control Page Flow Using the Wizard](#)
- [Tutorial: Using Page Flows Inside Portlets](#)

What You Need to Know

This document is intended for new or existing BEA partners who want to develop portals with the WebLogic Workshop Portal Extensions.

Product Documentation on the dev2dev Web Site

BEA product documentation, along with other information about BEA software, is available from the BEA dev2dev Web site:

<http://dev2dev.bea.com>

To view the documentation for a particular product, select that product from the list on the dev2dev page; the home page for the specified product is displayed. From the menu on the left

side of the screen, select Documentation for the appropriate release. The home page for the complete documentation set for the product and release you have selected is displayed.

Related Information

Readers of this document may find the following documentation and resources especially useful:

- For helpful information about programming with the Studio client, see the following books in the WebLogic Integration document set:
 - *Using the WebLogic Integration Studio*
 - *BEA WebLogic Integration Javadoc*
- For general information about Java applications, go to the Sun Microsystems, Inc. Java Web site at <http://java.sun.com>.
- For general information about XML, go to the O'Reilly & Associates, Inc. XML.com Web site at <http://www.xml.com>.

Contact Us!

Your feedback on the BEA Portal Tutorials documentation is important to us. Send us e-mail at **docsupport@bea.com** if you have questions or comments. Your comments will be reviewed directly by the BEA professionals who create and update the Portal Tutorials documentation.

In your e-mail message, please indicate that you are using the documentation for BEA Portal Tutorials ProductVersion.

If you have any questions about this version of BEA Portal Tutorials, or if you have problems installing and running BEA Portal Tutorials, contact BEA Customer Support at <http://support.bea.com>. You can also contact Customer Support by using the contact information provided on the quick reference sheet titled “BEA Customer Support,” which is included in the product package.

When contacting Customer Support, be prepared to provide the following information:

- Your name, e-mail address, phone number, and fax number
- Your company name and company address
- Your machine type and authorization codes
- The name and version of the product you are using

- A description of the problem and the content of pertinent error messages

Documentation Conventions

The following documentation conventions are used throughout this document.

Convention	Item
Ctrl+Tab	Indicates that you must press two or more keys simultaneously.
<i>italics</i>	Indicates emphasis or book titles.
monospace text	<p>Indicates <i>user input</i>, as shown in the following examples:</p> <ul style="list-style-type: none"> • Filenames: <code>config.xml</code> • Pathnames: <code>BEAHOME/config/examples</code> • Commands: <code>java -Dbea.home=BEA_HOME</code> • Code: <code>public TextMsg createTextMsg(</code> <hr/> <p>Indicates <i>computer output</i>, such as error messages, as shown in the following example:</p> <pre>Exception occurred during event dispatching:java.lang.ArrayIndexOutOfBoundsException: No such child: 0</pre>
monospace boldface text	<p>Identifies significant words in code.</p> <p><i>Example:</i></p> <pre>void commit ()</pre>
<i>monospace italic text</i>	<p>Identifies variables in code.</p> <p><i>Example:</i></p> <pre>String <i>expr</i></pre>
{ }	Indicates a set of choices in a syntax line. The braces themselves should never be typed.
[]	<p>Indicates optional items in a syntax line. The brackets themselves should never be typed.</p> <p><i>Example:</i></p> <pre>java utils.MulticastTest -n <i>name</i> [-p <i>portnumber</i>]</pre>
	<p>Separates mutually exclusive choices in a syntax line. The symbol itself should never be typed.</p> <p><i>Example:</i></p> <pre>java weblogic.deploy [<i>list deploy update</i>]</pre>

About This Document

Convention	Item
...	<p>Indicates one of the following in a command line:</p> <ul style="list-style-type: none">• That an argument can be repeated several times in a command line• That the statement omits additional optional arguments• That you can enter additional parameters, values, or other information <p>The ellipsis itself should never be typed.</p> <p><i>Example:</i></p> <pre>buildobjclient [-v] [-o name] [-f "file1.cpp file2.cpp file3.cpp . . ."]</pre>
.	<p>Indicates the omission of items from a code example or from a syntax line. The vertical ellipsis itself should never be typed.</p>

Portal Tutorials

The following tutorials highlight the processes and features of portal development with the WebLogic Workshop Portal Extensions:

- [Tutorial: Building Your First Portal](#)
- [Tutorial: Changing a Portal's Look & Feel and Navigation](#)
- [Tutorial: Showing Personalized Content in a Portlet](#)
- [Tutorial: Creating a Login Control Page Flow Using the Wizard](#)
- [Tutorial: Using Page Flows Inside Portlets](#)

Tutorial: Building Your First Portal

This tutorial guides you through the brief process of creating a portal with working portlets. The tutorial takes about 15 minutes to complete.

Tutorial Goals

At the end of this tutorial you will have built a fully functional portal in a short time with little effort.

Tutorial Overview

The WebLogic Workshop Portal Extensions include a graphical Portal Designer that lets you surface application functionality easily and quickly in a sophisticated portal interface. Sample portlets included with the WebLogic Workshop Portal Extensions provide instant, reusable functionality for a portal, as this tutorial illustrates.

Steps in This Tutorial

- [Step 1: Start WebLogic Workshop, Open an Application, and Start WebLogic Server – 5 minutes](#)

In this step you start WebLogic Workshop, open the sample portal application, and start WebLogic Server.

- [Step 2: Create a Portal](#) – 10 minutes

In this step you create a portal file for an existing portal project, add a page to the portal, add portlets to the pages, and view your finished portal.

Step 1: Start WebLogic Workshop, Open an Application, and Start WebLogic Server

In this step, you will start the development environment for building a portal.

The tasks in this step are:

- [Start WebLogic Workshop](#)
- [Open the Sample Portal Application](#)
- [Start WebLogic Server](#)

Start WebLogic Workshop

- **Windows** – Choose **Start**→**Programs**→**BEA WebLogic Platform 8.1**→**WebLogic Workshop**. The start script is located in the <BEA_HOME>/weblogic81/workshop directory.

Open the Sample Portal Application

The portal application you open contains all the necessary portal resources to complete the tutorial. In Java 2 Enterprise Edition (J2EE) terms, this application is an enterprise application that contains Web applications and related resources. Each Web application can contain multiple portals.

1. In the WebLogic Workshop menu, choose **File**→**Open**→**Application**.
2. In the Open Workshop Application dialog, select the <BEA_HOME>\weblogic81\samples\portal\portalApp\portalApp.work file and click **Open**.

The portalApp application directory tree appears in the Application window.

Start WebLogic Server

To develop portals and portal applications in WebLogic Workshop, WebLogic Server must be running on your development machine. For this tutorial, you will start the domain server used by the WebLogic Portal samples. The portalApp application you opened in the previous step contains all the necessary server configuration settings. To start WebLogic Server:

- In the WebLogic Workshop menu, choose **Tools**→**WebLogic Server**→**Start WebLogic Server**.

On Windows systems, you can bring up the command window from the Windows task bar to watch the startup progress. When the server starts, the WebLogic Workshop status bar shows the message "Server Running."

Step 2: Create a Portal

In this step, you will create a portal file for an existing portal project, add a page to the portal, add portlets to the pages, and view your finished portal.

The tasks in this step are:

- [Create a Portal File](#)
- [Add a Page to the Portal](#)
- [Add a Portlet to the Portal](#)
- [View the Portal](#)

Create a Portal File

A portal file is an XML file that contains all configuration information for a portal. The XML file is rendered graphically in the Portal Designer of the WebLogic Workshop Portal Extensions. As you build the portal in the graphical interface, the XML is generated automatically behind the scenes. To create a portal file:

1. In the **Application** window, right-click the **sampleportal** project and choose **New**→**Portal**.
2. In the **New File** dialog, enter my.portal in the **File name** field. You must keep the file extension.
3. Click **Create**.

The new file is added to the sampleportal project, and a new portal file appears in the Portal Designer. The new portal has a header, footer, and a main body containing a default book and page. The Document Structure window displays the component hierarchy.

Add a Page to the Portal

A new portal already contains a page. In this step you will add a second page to your portal and rename the tabs of both pages.

1. In the **Palette** window, drag the **Page** control into the Portal Designer next to the existing page tab. A page tab called "New Page" appears.
2. Click the **New Page** tab.
3. In the **Property Editor** window, change the page's **Title** property to My Page 2.
4. In the **Portal Designer**, click the **Page 1** tab.
5. In the **Property Editor** window, change the page's **Title** property to My Page 1 and press **Enter**.

Add a Portlet to the Portal

In this step you will add a pre-built sample portlet from the sampleportal project to each page.

1. In the Portal Designer, click the **My Page 1** tab.
2. In the **Data Palette** window, drag the **Login to Portal** portlet into the left placeholder of **My Page 1**.
3. In the **Portal Designer**, click the **My Page 2** tab.
4. In the **Data Palette** window, drag the **RSS News Feed** into the left placeholder of **My Page 2** and the **Dev2Dev** portlet into the right placeholder.

Note: The RSS News Feed portlet requires an Internet connection to access the content feed.

5. Save the portal file.

View the Portal

You can view your portal with the WebLogic Test Browser or with your default browser.

- **WebLogic Test Browser** – In the WebLogic Workshop toolbar, click the **Start** button (or press **Ctrl+F5**). Navigate between the portal pages using the **My Page 1** and **My Page 2** links.
- **Default Browser** – In the WebLogic Workshop menu, choose **Portal**→**Open Current Portal**. Navigate between the portal pages using the **My Page 1** and **My Page 2** links.

Congratulations! You have built a portal.

What You Can Do Next

The portal development life cycle involves development with the WebLogic Workshop Portal Extensions and administration with the WebLogic Administration Portal. The tutorial you have just completed covers the development phase. The following steps instructions for starting a tutorial that covers the administration phase. The second phase of the portal-building process involves administering the portal you have created. To run a companion tutorial for administering a portal:

1. In the WebLogic Workshop menu, choose **Portal**→**Open Portal Administration**.
2. Log in with Username: **weblogic** Password: **weblogic**.
3. When the Administration Portal appears, click **Show Help** in the upper left corner of the window.
4. In the Help window, click the **Tutorials** tab, select **Build Your First Portal**, and follow the tutorial.

Tutorial: Changing a Portal's Look & Feel and Navigation

This tutorial shows you how easy it is to modify the look and feel of a portal and modify the navigation style used for the portal pages. The tutorial takes about 10 minutes to complete.

Tutorial Goals

At the end of this tutorial you will have changed the look and feel and page navigation style of a portal.

Tutorial Overview

This tutorial involves modifying two elements of a portal's physical appearance and behavior: look and feel, and navigation.

The WebLogic Workshop Portal Extensions provide a flexible, extensible architecture for controlling the look and feel and page navigation in a portal. A portal look and feel is made up of a skin (graphics, a cascading style sheet, and JavaScript functions) and skeletons (JSP files that determine the rendering—the physical boundaries—of individual portal components, such as desktops, pages, and portlets).

Ultimately the look and feel of a portal, and its navigation style, are determined by the portal administrator and end users. All look and feel and navigation resources that are available to developers in the WebLogic Workshop Portal Extensions are also available to delegated administrators in the WebLogic Administration Portal and to end users in the Visitor Tools when the portal is put into production.

When an administrator creates a desktop based on a .portal file in the WebLogic Administration Portal, that portal is decoupled from the development environment and can be modified as needed by the administrator, and in turn by end users when the portal is put into production. The initial look and feel and navigation settings you provide in development serve as the default settings for portal administrators and end users. This tutorial merely shows how easy it is to change look and feel and navigation elements in the development environment.

Steps in This Tutorial

- [Step 1: Start WebLogic Workshop, Open an Application, and Start WebLogic Server](#) – 5 minutes

In this step you start WebLogic Workshop, open the sample portal application, and start WebLogic Server.

- [Step 2: Change the Portal Look & Feel and Navigation](#) – 5 minutes

In this step you change a portal's look and feel and page navigation style.

Step 1: Start WebLogic Workshop, Open an Application, and Start WebLogic Server

In this step, you will start the development environment for working with portals.

The tasks in this step are:

- [Start WebLogic Workshop](#)
- [Open the Sample Portal Application](#)
- [Start WebLogic Server](#)

Start WebLogic Workshop

- **Windows** – Choose **Start**→**Programs**→**BEA WebLogic Platform 8.1**→**WebLogic Workshop**. The start script is located in the <BEA_HOME>/weblogic81/workshop directory.

Open the Sample Portal Application

The portal application you open contains all the necessary portal resources to complete the tutorial. In Java 2 Enterprise Edition (J2EE) terms, this application is an enterprise application that contains Web applications and related resources.

1. In the WebLogic Workshop menu, choose **File**→**Open**→**Application**.
2. In the Open **Workshop Application** dialog, select the <BEA_HOME>\weblogic81\samples\portal\portalApp\portalApp.work file and click **Open**.

The portalApp application directory tree appears in the **Application** window.

Start WebLogic Server

To develop portals and portal applications in WebLogic Workshop, WebLogic Server must be running on your development machine. For this tutorial, you will start the domain server used by the WebLogic Portal samples. The portalApp application you opened in the previous step contains all the necessary server configuration settings. To start WebLogic Server:

- In the WebLogic Workshop menu, choose **Tools**→**WebLogic Server**→**Start WebLogic Server**.

On Windows systems, you can bring up the command window from the Windows task bar to watch the startup progress. When the server starts, the WebLogic Workshop status bar shows the message "Server Running."

Step 2: Change the Portal Look & Feel and Navigation

In this step, you will change a portal's look and feel, change the navigation scheme for the portal pages, and view the results.

The tasks in this step are:

- [Open the Sample Portal](#)
- [View the Portal in a Browser](#)
- [Add a Book to the Main Page Book](#)
- [Change the Portal's Look and Feel](#)
- [Change the Portal's Navigation](#)
- [View the Modified Portal in a Browser](#)

Open the Sample Portal

The Sample portal is included with the WebLogic Workshop Portal Extensions. It contains a set of pre-built sample portlets you can reuse in your own portals. You will change the Sample portal's look and feel and page navigation scheme.

1. In the **Application** window, expand the **sampleportal** folder.
2. Double-click **sample.portal**. The portal file opens in the Portal Designer.

View the Portal in a Browser

In this step, view the existing look and feel and navigation of the Sample portal to see what it looks like before you change it.

1. In the WebLogic Workshop toolbar, click the **Start** button (or press **Ctrl+F5**) to view the portal in the Workshop Test Browser.

Click the page navigation tabs and note the behavior.

2. Close the WebLogic Test Browser.

Add a Book to the Main Page Book

1. In the **Palette** window, drag the **Book** control into the Main Page Book area of the Portal Designer, next to **My Page**. A new book appears in the main page book.

Do NOT drag the Book control into a placeholder on one of the pages.

2. Click the **New Book** tab. Notice that it automatically contains a new page as well.

Change the Portal's Look and Feel

1. In the **Document Structure** window, click **Desktop**.
2. In the **Property Editor** window, change the **Look and Feel** property from **avitek** to **Classic**.

Change the Portal's Navigation

1. In the **Document Structure** window, click **Book: Main Page Book**.
2. In the **Property Editor** window, change the Navigation property from **Single Level Menu** to **Multi Level Menu**. Multi-level menus display links to nested books and pages with drop-down navigation.
3. Save the portal file.

View the Modified Portal in a Browser

You can view the portal with the WebLogic Test Browser or with your default browser.

- **WebLogic Test Browser** – In the WebLogic Workshop toolbar, click the **Start** button (or press **Ctrl+F5**).
- **Default Browser** – In the WebLogic Workshop menu, choose **Portal→Open Current Portal**.

Use the page and book navigation links and note the different navigation behavior. The New Book link should provide a drop-down navigation menu to access the page it contains.

Congratulations! You have changed a portals look and feel and page navigation style.

Tutorial: Showing Personalized Content in a Portlet

This tutorial has you develop personalization functionality and surface that functionality in a portlet. The tutorial takes about 30 minutes to complete.

Tutorial Goals

At the end of this tutorial you will have created a portlet that displays different personalized content to different users.

Tutorial Overview

The WebLogic Workshop Portal Extensions include a graphical Portal Designer that lets you create and surface application functionality easily and quickly in a sophisticated portal interface. In this tutorial you will use the WebLogic Workshop Portal Extensions and the WebLogic Administration Portal to create the users, properties, content, and code that results in a complete personalization solution. You will perform minimal JSP development using Portal JSP tags in an easy-to-use, integrated interface.

Steps in This Tutorial

- [Step 1: Start WebLogic Workshop, Open an Application, and Start WebLogic Server](#) – 5 minutes

In this step you start WebLogic Workshop, open the sample portal application, and start WebLogic Server.

- [Step 2: Create a User Profile Property Set](#) – 5 minutes

In this step you will create the user properties that uniquely identify authenticated users and determine what the users see.

- [Step 3: Create Two Users](#) – 5 minutes

In this step you will create two test users and assign property values to them.

- [Step 4: Load Content](#) – 5 minutes

In this step you will load content into the BEA Virtual Content Repository.

- [Step 5: Create Two Content Selectors](#) – 5 minutes

In this step you will create two content selectors, each of which will provide personalized content to different types of users.

- [Step 6: Create a Portlet](#) – 3 minutes

In this step you will create a portlet and a JSP by simply dragging and dropping a content selector into a portal.

Step 1: Start WebLogic Workshop, Open an Application, and Start WebLogic Server

- [Step 7: Modify the JSP](#) – 3 minutes

In this step you will open the new JSP, view the automatically generated code, and add the second content selector code to the JSP.

- [Step 8: Test the Personalized Portlet](#) – 5 minutes

In this step you will log in as both users to see the personalized portlet in action.

Step 1: Start WebLogic Workshop, Open an Application, and Start WebLogic Server

In this step, you will start the development environment for working with portals.

The tasks in this step are:

- [Start WebLogic Workshop](#)
- [Open the Sample Portal Application](#)
- [Start WebLogic Server](#)

Start WebLogic Workshop

- **Windows** – Choose **Start**→**Programs**→**BEA WebLogic Platform 8.1**→**WebLogic Workshop**. The start script is located in the <BEA_HOME>/weblogic81/workshop directory.

Open the Sample Portal Application

The portal application you open contains all the necessary portal resources to complete the tutorial. In Java 2 Enterprise Edition (J2EE) terms, this application is an enterprise application that contains Web applications and related resources.

1. In the WebLogic Workshop menu, choose **File**→**Open**→**Application**.
2. In the Open **Workshop Application** dialog, select the <BEA_HOME>\weblogic81\samples\portal\portalApp\portalApp.work file and click **Open**.

The portalApp application directory tree appears in the **Application** window.

Start WebLogic Server

To develop portals and portal applications in WebLogic Workshop, WebLogic Server must be running on your development machine. For this tutorial, you will start the domain server used by the WebLogic Portal samples. The portalApp application you opened in the previous step contains all the necessary server configuration settings. To start WebLogic Server:

- In the WebLogic Workshop menu, choose **Tools**→**WebLogic Server**→**Start WebLogic Server**.

On Windows systems, you can bring up the command window from the Windows task bar to watch the startup progress. When the server starts, the WebLogic Workshop status bar shows the message "Server Running."

Step 2: Create a User Profile Property Set

In this step, you will create a user profile property set. This property set will contain properties that can be set for all users. Each user can have different property values. In this tutorial, property values determine the personalized content users see.

The tasks in this step are:

- [Create a Property Set File](#)
- [Add a Property to the Property Set](#)

Create a Property Set File

1. In the **Application** window, expand the **data** project.
2. Right-click the **userprofiles** folder, and choose **New**→**User Profile Property Set**.
3. In the **New File** dialog, enter userpreferences.usr in the **File name** field. You must keep the file extension.
4. Click **Create**. The **Property Set Designer** appears.

Add a Property to the Property Set

1. In the **Palette** window, drag the **Single Restricted** icon into the Property Set Designer.
2. In the **Property Editor** window, enter the following text in the **Property Name** field:
Graphic Preference.

3. Click the ellipsis icon [...] in the **Value(s)** field.

4. In the **Enter Property Value** dialog:

Enter **modern** in the top field and click **Add**.

Enter **classic** in the top field and click **Add**.

Enter **other** in the top field and click **Add**.

Select **other** as the default property.

5. Click **OK**.

6. Save and close the property set file.

Step 3: Create Two Users

In this step, you will create two test users that have different preferences set in their user profiles. When you log in as each user at the end of the tutorial, each users will see different content displayed in the portlet based on their different user profile property values.

The tasks in this step are:

- [Start the WebLogic Administration Portal](#)
- [Create Two Users and Set Their Profile Preferences](#)

Start the WebLogic Administration Portal

1. In the WebLogic Workshop menu, choose **Portal**→**Open Portal Administration**.
2. Log in with Username: **weblogic** Password: **weblogic**.
3. When the WebLogic Administration Portal appears, select **Users & Groups** under Users, Groups, & Roles in the top Menu in the editor pane.

Create Two Users and Set Their Profile Preferences

1. In the Users & Groups resource tree, click **everyone (All Users)**.
2. Select the **Add Users** page in the editor pane.
3. Click **Create New User** in the main window.

4. In the Add a New User dialog that appears, enter `modernuser`, enter a password of `password` (which is the default), and click **Create New User**. A confirmation message appears at the top of the Add Users page.
5. Create a second user called `classicuser` with a password of `password`.
6. Select the **Edit Users** page.
7. In section **1** of the page, enter an asterisk (*) in the Search field and click **Search**.
8. In section **2** of the page, click the `classicuser` name. The **Editing User** window appears.
9. Select the **Edit User Profile Values** page.
10. In the **Properties from property set** field, select `userpreferences`. This is the property set you created.
11. Expand the **Graphic Preference** property line. Change the Graphic Preference property value to `classic`, and click **Update Value**.
12. In the left resource tree, select `everyone (All Users)`.
13. In section **1** of the page, enter an asterisk (*) in the Search field and click **Search**.
14. In section **2** of the page, click the `modernuser` name. The **Editing User** window appears.
15. On the **Edit User Profile Values** page, change the user's Graphic Preference to `modern`, and click **Update Value**.

You now have two users with different user profile preferences.

Step 4: Load Content

In this step, you will load sample content into the Virtual Content Repository that will be used later in this tutorial. This step requires WebLogic Server to be running, as described in Step #1.

1. Open a command window and change to the following directory:

```
<BEA_HOME>\weblogic81\portal\bin
```

2. Enter the following command:

```
load_cm_data.cmd  
or  
sh load_cm_data.sh
```


The script loads sample content into the sample domain's Virtual Content Repository under the default "BEA Repository." You can view this sample content in the WebLogic Administration Portal using the following steps:

1. Choose **Portal**→**Open Portal Administration** in WebLogic Workshop or by entering **http://localhost:7001/portalAppAdmin** in a browser.
2. Log in as weblogic/weblogic.
3. In the WebLogic Administration Portal, select **Content** in the top Menu in the editor pane.
4. In the left resource tree, expand the BEA Repository.

Step 5: Create Two Content Selectors

In this step, you will create two content selectors that contain the queries for retrieving content from the BEA Virtual Content Repository and the rules that trigger the queries to run.

The tasks in this step are:

- [Create a "modern" content selector](#)
- [Create a "classic" content selector](#)

Create a "modern" content selector

1. In the WebLogic Workshop Application window, right-click **data\contentselectors\GlobalContentSelectors**, and choose **New**→**Content Selector**.
2. In the New File window, enter **modern.sel** for the content selector name, and click **Create**.
3. In the Content Selector Designer, click the **[empty content search]** link. The Content Search window appears.
4. In the Property set field, select **Standard**.
5. In the Property field, make sure **cm_binaryName** is selected, and click **Add**. The Content Search Values window appears.
6. In the Comparison field, select **contains**.
7. In the Value field, enter **college** and click **Add**.
8. Click **OK**. You are returned to the Content Search window. Click **OK**.

9. In the Available Conditions area of the Content Selector designer, right-click "The current date is after," and choose **Delete**.
10. In the Palette window, drag **The visitor has specific characteristics** into the Content Selector designer.
11. In the top of the Content Selector Designer, click the **[characteristics]** link. The Visitor Characteristics window appears.
12. In the Visitor property set field, select **userpreferences**.
13. In the Visitor property field, make sure **Graphic Preference** is selected, and click **Add**. The Visitor Characteristic Values window appears.
14. In the Comparison field, make sure **is equal to** is selected.
15. In the Value field, select **modern**, click **Add**, and click **OK**.
16. In the Visitor Characteristics window, click **OK**.
17. Save and close the file.

Create a "classic" content selector

1. Create another content selector called **classic.sel**.
2. In the Content Selector Designer, click the **[empty content search]** link. The Content Search window appears.
3. In the Property set field, select **Standard**.
4. In the Property field, make sure **cm_binaryName** is selected, and click **Add**. The Content Search Values window appears.
5. In the Comparison field, select **contains**.
6. In the Value field, enter **IRACampaign** and click **Add**.
7. Click **OK**. You are returned to the Content Search window. Click **OK**.
8. In the Available Conditions area of the Content Selector designer, right-click "The current date is after," and choose **Delete**.
9. In the Palette window, drag **The visitor has specific characteristics** into the Content Selector designer.

10. In the top of the Content Selector Designer, click the [**characteristics**] link. The Visitor Characteristics window appears.
11. In the Visitor property set field, select **userpreferences**.
12. In the Visitor property field, make sure **Graphic Preference** is selected, and click **Add**. The Visitor Characteristic Values window appears.
13. In the Comparison field, make sure **is equal to** is selected.
14. In the Value field, select **classic**, click **Add**, and click **OK**. In the Visitor Characteristics window, click **OK**. Save and close the file.
15. In the Visitor Characteristics window, click **OK**. Save and close the file.
16. Save and close the file.

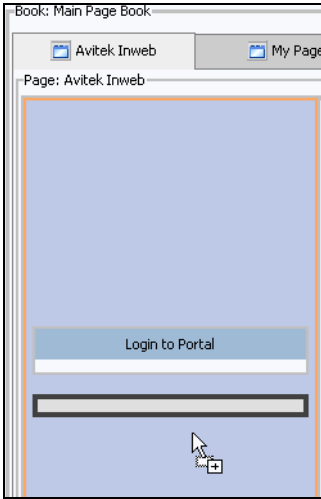
Step 6: Create a Portlet

In this step, you will create a portlet out of the content selector. During this process, a Java Server Page (JSP) will automatically be created that will be used to show the personalized content in the portlet.

Create a Portlet

1. In WebLogic Workshop, expand the **sampleportal** project in the Application window and choose double-click **sample.portal** to open the portal file.
2. In the Application window, expand the **data\contentselectors\GlobalContentSelectors** directory.

3. Drag **modern.sel** into the portal just below the Login to Portal portlet.



4. When prompted to create a portlet, click **Yes**.
5. In the Portlet Wizard that appears, enter the following:

Title: My Personalized Portlet
Content URI: /myp13n.jsp

6. Click **Finish**.
7. In the Generate Files window that appears, click **OK**.
8. Save the `sample.portal` file.

The new portlet is added below the Login to Portal portlet, and the following resources have been created in the `\sampleportal` directory: `modern.portlet` and `myp13n.jsp`.

In the next step, you will open the new JSP, view the code that was generated for the modern content selector, and add code to include the classic content selector.

Step 7: Modify the JSP

In this step, you will open the new `myp13n.jsp`, view the code that was automatically generated for the modern content selector, and add the classic content selector to the JSP.

1. In the Portal Designer, double-click the new My Personalized Portlet. The modern.portlet file is opened in the Portlet designer.
2. Double-click the portlet in the Portlet designer. The new myp13n.jsp is opened in the JSP designer.
3. In the JSP designer, click the **Source View** tab. The following code should appear:

```
<!--Generated by WebLogic Workshop-->
<%@ page language="java" contentType="text/html; charset=UTF-8"%>
<%@taglib uri="http://www.bea.com/servers/portal/tags/personalization"
prefix="pz"%>
<%@taglib prefix="utility"
uri="http://www.bea.com/servers/p13n/tags/utility"%>
<%@taglib prefix="cm" uri="content.tld"%>

<!-- Retrieve the content and display each node name in a list. -->
<pz:contentSelector rule="modern" id="nodes"/>
<utility:NotNull item="<%=nodes%>">
<ul>
<utility:forEachInArray array="<%=nodes%>" id="node"
type="com.bea.content.Node">
<li><cm:getProperty id="node" name="cm_nodeName"
conversionType="html"/></li>
</utility:forEachInArray>
</ul>
</utility:NotNull>
```

Notice that several JSP tags have been generated, along with their necessary tag library include statements at the top.

4. Delete the following code:

```
<li><cm:getProperty id="node" name="cm_nodeName"
conversionType="html"/></li>
```

and replace it with this:

```
">
```

5. Copy the entire code block from <!-- Retrieve the content...> to the closing </utility:NotNull> tag, and paste the new code block below the block you just copied.
6. In the newly pasted code block, change the **rule** attribute in the <pz:contentSelector> tag to **rule="classic"**. The JSP should now look like this:

```
<%--Generated by WebLogic Workshop--%>
<@ page language="java" contentType="text/html; charset=UTF-8"%>
<@taglib uri="http://www.bea.com/servers/portal/tags/personalization"
prefix="pz"%>
<@taglib prefix="utility"
uri="http://www.bea.com/servers/p13n/tags/utility"%>
<@taglib prefix="cm" uri="content.tld"%>

<%-- Retrieve the content and display each node name in a list. --%>
<pz:contentSelector rule="modern" id="nodes"/>
<utility:notNull item="<%=nodes%>">
<ul>
<utility:forEachInArray array="<%=nodes%>" id="node"
type="com.bea.content.Node">
">
</utility:forEachInArray>
</ul>
</utility:notNull>

<%-- Retrieve the content and display each node name in a list. --%>
<pz:contentSelector rule="classic" id="nodes"/>
<utility:notNull item="<%=nodes%>">
<ul>
<utility:forEachInArray array="<%=nodes%>" id="node"
type="com.bea.content.Node">
">
</utility:forEachInArray>
</ul>
</utility:notNull>
```

The JSP now contains both content selectors. Save and close the JSP file. Close the portlet file.

7. Save and close the JSP file.
8. Close the portlet file.

Step 8: Test the Personalized Portlet

In this step, you will test the portlet to see personalization in action.

The tasks in this step are:

- [View the Sample Portal](#)

- [Log in as One User and View the Portlet](#)
- [Log in as the Other User and View the Portlet](#)

View the Sample Portal

View the portal containing the new personalized portlet. In the WebLogic Workshop menu, choose **Portal**→**Open Current Portal**.

When the portal appears in the browser, the My Personalized Content portlet should not display content.

Log in as One User and View the Portlet

1. In the **Login** portlet, log in with **Username**: classicuser **Password**: password.
2. A personalized image should appear in the portlet.
3. Click **Logout**. The personalized content disappears from the portlet.

Log in as the Other User and View the Portlet

1. Log in with **Username**: modernuser **Password**: password.
 2. A different personalized image should appear in the portlet.
- Congratulations! You have created a portlet that uses personalization.

Tutorial: Creating a Login Control Page Flow Using the Wizard

This tutorial shows you how easy it is to add a Portal Control into a Page Flow. The tutorial takes about 25 minutes to complete.

At the end of this tutorial you will have created a Page Flow that includes a Portal Control, and that can be placed inside a portlet to expose the login functionality.

Tutorial Overview

This tutorial introduces the use of Portal Controls with Page Flows, and should help familiarize you with databinding in forms.

Note: For instructions on adding the User Login Control without the wizard, consult the User Login Control help.

Steps in This Tutorial

- [Step 1: Start WebLogic Workshop, Open an Application, and Start WebLogic Server](#) – 5 minutes

In this step you start WebLogic Workshop, open the sample portal application, and start WebLogic Server.

- [Step 2: Create and Configure a Page Flow](#) – 10 minutes

In this step you create a new Page Flow, which includes selecting the Login Control.

- [Step 3: Create a Portlet and Edit the Login Form](#) – 10 minutes

In this step the new Page Flow is used to create a new portlet.

Step 1: Start WebLogic Workshop, Open an Application, and Start WebLogic Server

In this step, you will start the development environment for building a portal.

The tasks in this step are:

- [Start WebLogic Workshop](#)
- [Open the Sample Portal Application](#)
- [Start WebLogic Server](#)

Start WebLogic Workshop

- **Windows** – Choose **Start**→**Programs**→**BEA WebLogic Platform 8.1**→**WebLogic Workshop**. The start script is located in the <BEA_HOME>/weblogic81/workshop directory.

Open the Sample Portal Application

The portal application you open contains all the necessary portal resources to complete the tutorial. In Java 2 Enterprise Edition (J2EE) terms, this application is an enterprise application that contains Web applications and related resources. Each Web application can contain multiple portals.

1. In the WebLogic Workshop menu, choose **File**→**Open**→**Application**.

2. In the Open **Workshop Application** dialog, select the `<BEA_HOME>\weblogic81\samples\portal\portalApp\portalApp.work` file and click **Open**.

The portalApp application directory tree appears in the **Application** window.

Start WebLogic Server

To develop portals and portal applications in WebLogic Workshop, WebLogic Server must be running on your development machine. For this tutorial, you will start the domain server used by the WebLogic Portal samples. The portalApp application you opened in the previous step contains all the necessary server configuration settings. To start WebLogic Server:

- In the WebLogic Workshop menu, choose **Tools**→**WebLogic Server**→**Start WebLogic Server**.

On Windows systems, you can bring up the command window from the Windows task bar to watch the startup progress. When the server starts, the WebLogic Workshop status bar shows the message "Server Running."

Step 2: Create and Configure a Page Flow

In this step, you will create a Portal Control Page Flow using the Page Flow Wizard and then configure the page flow by directly modifying methods in Source View.

1. From a Portal Web application (such as **portalApp**) within WebLogic Workshop, create a new Page Flow by right-clicking on the portlets directory within the current Portal project (such as **samplePortal**) and selecting **New**→**Page Flow**.

2. The Page Flow Wizard appears. Name this new Page Flow "userLogin", and click **Next**.

Page Flow Wizard - Page Flow Name

Name And Location

Page Flow Name:

Location:

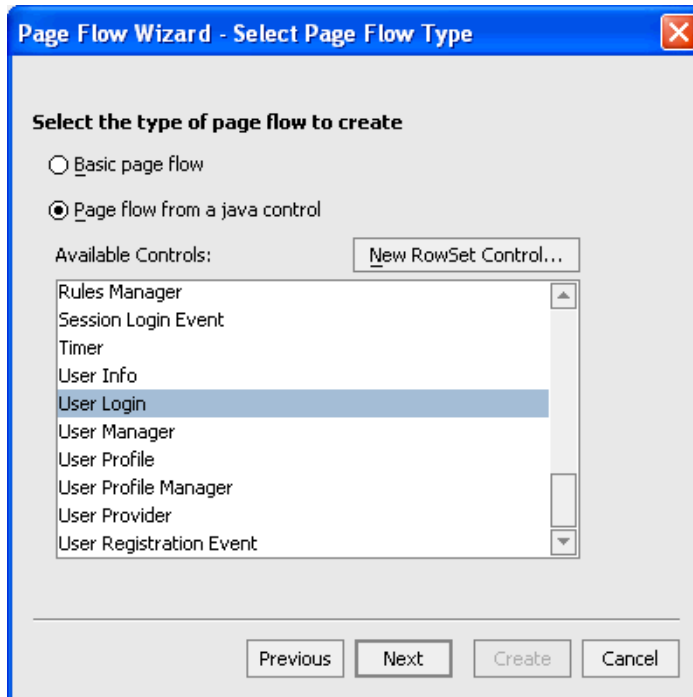
Controller Name:

Page Flow Nesting

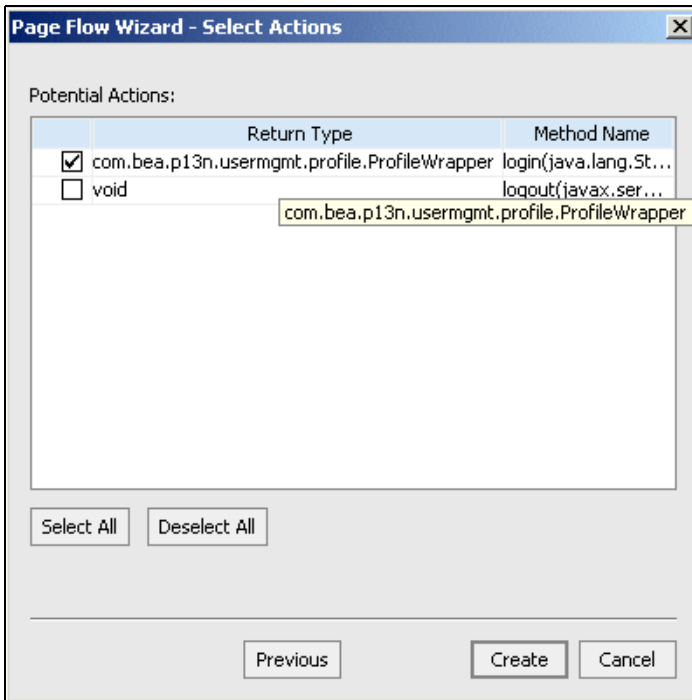
Nested page flows are used to gather and return information to a calling page flow.

Make this a nested page flow

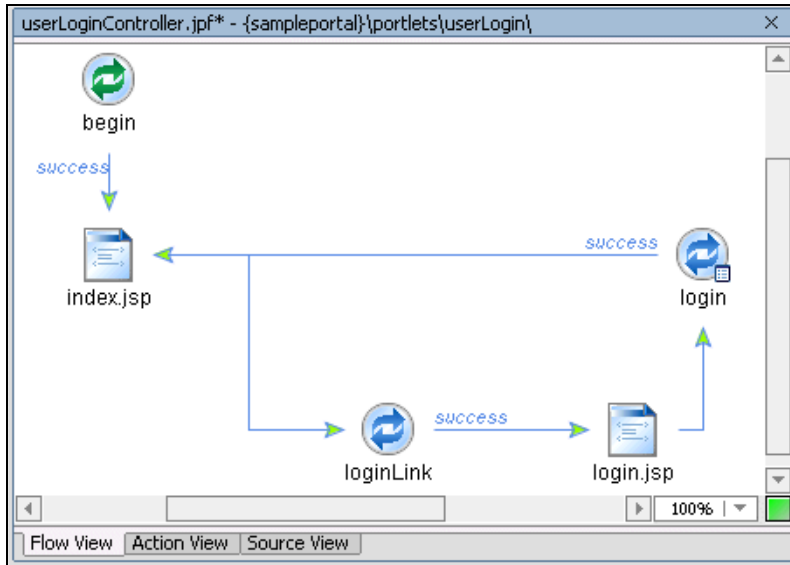
3. At the Select Page Flow Type prompt, select **Page flow from a Java Control**, select the **User Login**, and click **Next**.



- From the Select Actions prompt, select the **login()** method, which returns the ProfileWrapper type. Click **Create**.



5. The resulting Page Flow should appear in Flow View.



6. Press **Ctrl+S** to save your work. Notice that in addition to the **userLoginController** page flow file, the wizard has also generated **index.jsp** and **login.jsp**.

The wizard has created a page flow with all the necessary elements in place, but they must be configured to fit your application. The pageflow needs to be modified to pass in the "request" to the methods. By default, the pageflow will create a formbean property for every parameter.

7. Open the `userLoginController.jspf` in **Source View**, and within the control method `login`, replace the existing code with the code in red:

```
com.bea.p13n.usermgmt.profile.ProfileWrapper var = myControl.login(
aForm.username, aForm.password, super.getRequest() );
```

8. Now open the `userLoginController.jspf` in **Action View**, and from the Data Palette, open the control (it should be called **myControl**), and drag the **logout()** method into the Page Flow.

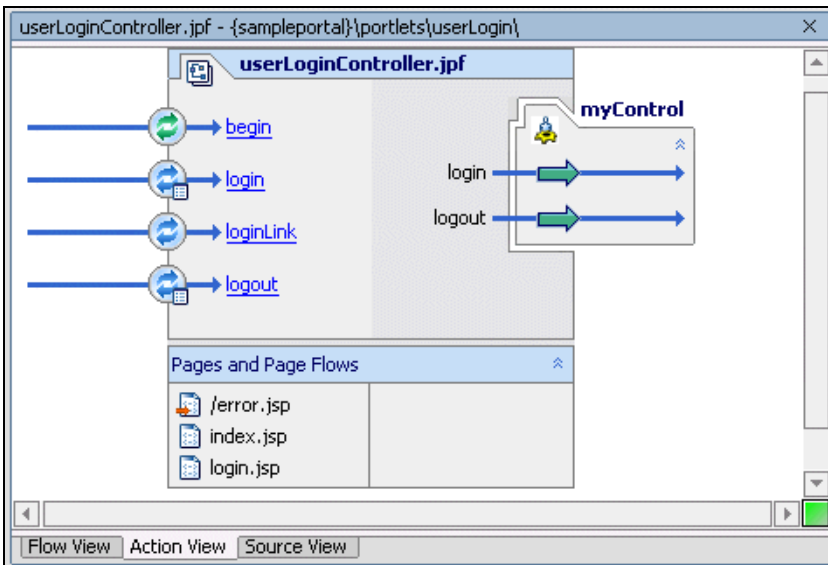
Double-clicking on the logout action will open the Source View to the logout method, shown below:

```
/**
 * @jpf:action
 */
protected Forward logout(LoginForm form)
{
    myControl.logout(form.getRequest());
    return new Forward( "success" );
}
```

9. Modify the logout method by adding the following code shown in red:

```
/**
 * @jpf:action
 * @jpf:forward name="success" path="index.jsp"
 */
protected Forward logout(LoginForm form)
{
    myControl.logout(this.getRequest());
    return new Forward( "success" );
}
```

10. From the Action View, the userLoginController.jsp should now look like this:



11. Press **Ctrl+S** to save your work.
12. Next, double-click on the **index.jsp** and open Source View. Just below the login form, add a logout button:

```
<netui:anchor action="loginLink">
  login
</netui:anchor>
<netui:form action="logout">
  <netui:button type="submit" value="logout"
    action="logout"></netui:button>
</netui:form>
```

The code in red above places a button of type "submit" in the bottom of the page, associates it with the **logout** action, wraps it inside a netui:form that invokes the action="logout". Save the index.jsp file.

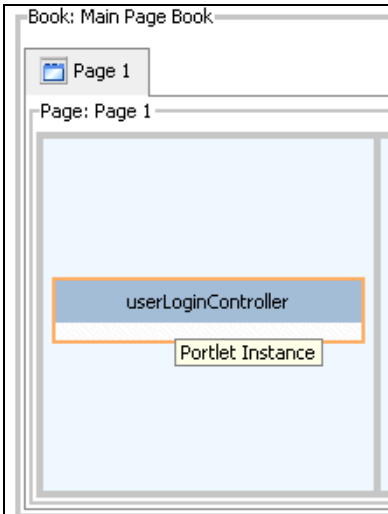
13. Save the **index.jsp** file.

Step 3: Create a Portlet and Edit the Login Form

In this step, you first create a Portlet from the new page flow and then edit the login form.

1. In the Portal Designer, drag the Page Flow file, called **userLoginController.jpf**, into a place holder in a portal.
2. In the Create Portlet dialog, click **Yes**.

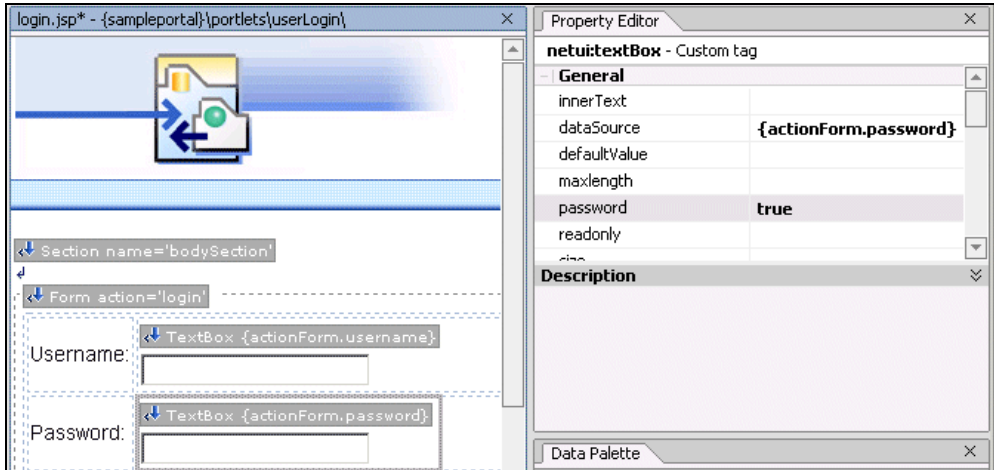
- From the Portlet Details screen, click **Finish** to create the **userLoginController** portlet and place it in the portal.



- Open **login.jsp** in Source view and make sure the Username text box is above the Password text box. The code that inserts these text boxes should look like this:

```
<tr class="tbody">
  <td>Username:</td>
  <td>
    <netui:textBox dataSource="{actionForm.username}"/>
  </td>
</tr>
<tr class="tbody">
  <td>Password:</td>
  <td>
    <netui:textBox dataSource="{actionForm.password}"
      password="true"/>
  </td>
</tr>
```


- Open **login.jsp** in the Design View and select the Password text box. Then, in the Properties Editor, scroll down the list of General Properties to set Password equal to **true**. This will cause any input in this field to be masked.



- For debugging purposes, the next step shows how to add some code to show which user is logged on. Open **index.jsp** in the Source View, adding the following code to the end, just before the last two lines:

```
<% if (request.getRemoteUser() != null) { %>
    <BR>you are logged in as: <%=request.getRemoteUser()%>
    <br>
    <%
}
else
{
    %>
    <BR>you are not logged in
<%
}
%>
```

The index.jsp should read as follows, with the inserted code shown in red:

```
<@ 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-template:templatetemplatePage="/resources/jsp/template.jsp">
  <netui-template:setAttribute value="Index" name="title"/>
  <netui-template:section name="bodySection">
    <p class="pagehead">
      &nbsp;&nbsp;&nbsp;Page Flow: yyyFlow
    </p>
    <table width="100%" cellpadding="0" class="tablebody"
      cellspacing="0">
      <tr>
        <td valign="top">
          <table width="100%" class="tablebody">
            <tr class="tablehead">
              <td>Actions With No Parameters</td>
            </tr>
          </table>
        </td>
        <td valign="top">
          <table width="100%" class="tablebody">
            <tr class="tablehead">
              <td>Input Forms For Actions With
                Parameters</td>
            </tr>
            <tr>
              <td>
                <netui:anchor action="loginLink">
                  login
                </netui:anchor>
                <netui:form action="logout">
                <netui:button type="submit" value="logout"
                  action="logout"></netui:button>
                </netui:form>
              </td>
            </tr>
          </table>
          <br/>
          <br/>
        </td>
      </tr>
      <tr class="tablehead">
        <td align="left" colspan="2">
          Results Area
        </td>
      </tr>
    </table>
  <br/>
  <%Object res = request.getAttribute ( "results" );%><%= (res

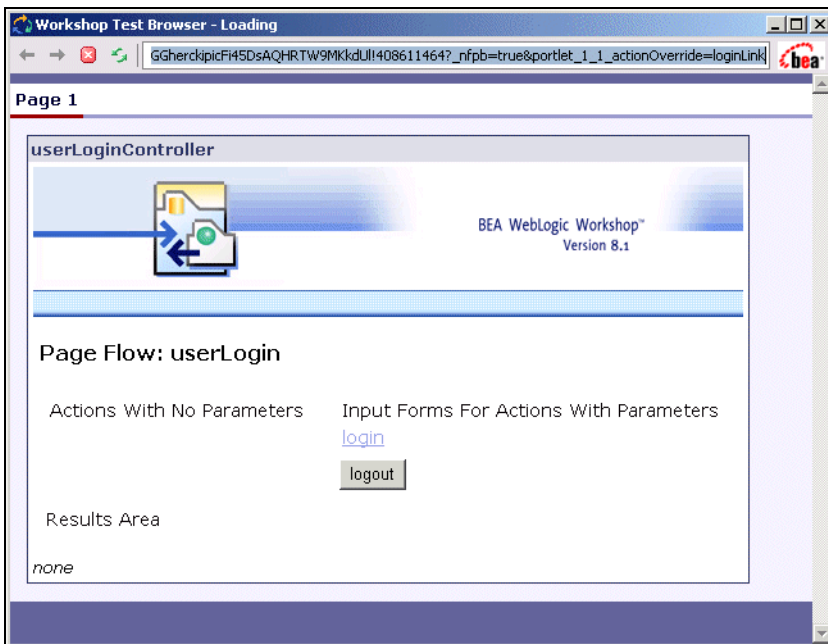
```

Step 3: Create a Portlet and Edit the Login Form

```
        == null ? "<i>none</i><br/>" : ( res + "<br/>"))%>
<% if (request.getRemoteUser() != null) { %>
    <BR>you are logged in as: <%=request.getRemoteUser()%>
    <br>
    <%
    }
else
{
    %>
    <BR>you are not logged in
    <%
    }
    %>
</netui-template:section>
</netui-template:template>
```

7. Click **Ctrl+S** to save your work. In the directory tree of the Application window, double-click the .portal file to give it focus. In the WebLogic Workshop menu, choose **Portal→Open Current Portal**.
8. In the directory tree of the Application window, double-click the .portal file to give it focus.


9. In the WebLogic Workshop menu, choose **Portal**→**Open Current Portal**.



10. Click on Login, and at the login page, enter **weblogic/weblogic**. (The password field should be masked.)

Page 1

userLoginController



BEA WebLogic Workshop™
Version 8.1

Username:


Password:

Request: *To-Do: To create an input form for a complex data type (`javax.servlet.http.HttpServletRequest`), open the page editor and add netui input tags for the relevant fields of the class.*

11. Verify that the next page displays the current logged-in user.

Page 1

userLoginController



BEA WebLogic Workshop™
Version 8.1

Page Flow: userLogin

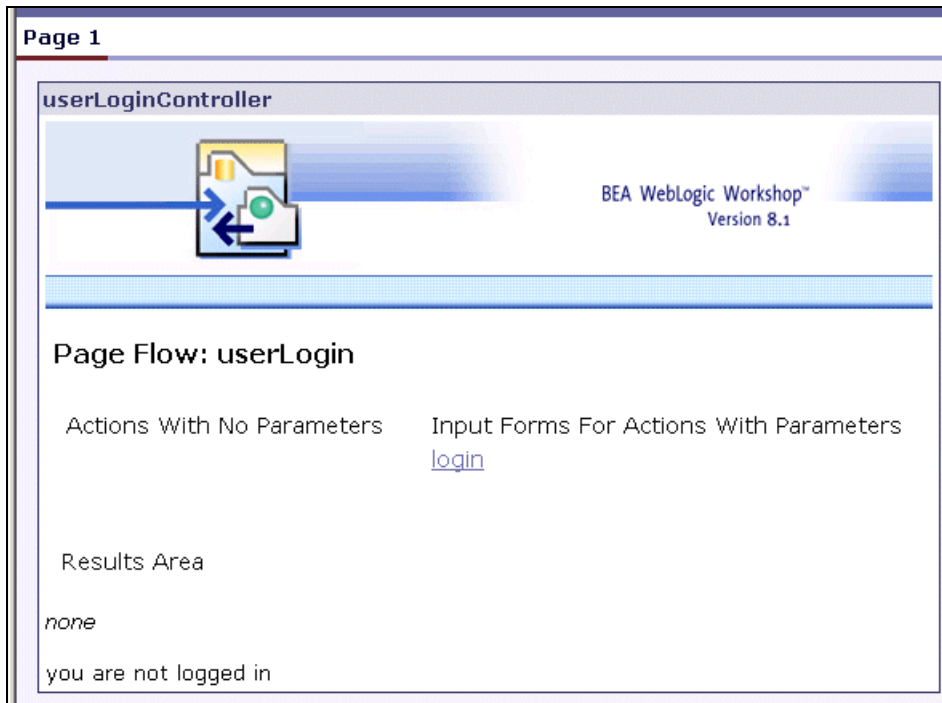
Actions With No Parameters	Input Forms For Actions With Parameters
	login
	<input type="button" value="logout"/>

Results Area

com.bea.p13n.usermgmt.profile.internal.ProfileWrapperImpl@1646436
[id=weblogic:null]

you are logged in as: weblogic

12. Click **logout**, and verify the page displays the "you are not logged in" message.



Congratulations! You have created a portlet that uses a Page Flow and an EJB control.

Tutorial: Using Page Flows Inside Portlets

This tutorial guides you through the process of learning how to use Page Flows inside portlets. The tutorial takes about 50 minutes to complete.

Tutorial Goals

At the end of this tutorial you will have some familiarity with how to use Page Flows inside portlets.

Note: Before starting this tutorial, you may wish to familiarize yourself with Page Flows by looking at the Page Flow tutorial first.

Tutorial Overview

The WebLogic Workshop Portal Extensions include a graphical Portal Designer that lets you surface application functionality easily and quickly in a sophisticated portal interface. Sample portlets included with the WebLogic Workshop Portal Extensions provide instant, reusable functionality for a portal, as this tutorial illustrates.

The portal development life cycle involves development with the WebLogic Workshop Portal Extensions and administration with the WebLogic Administration Portal. This tutorial covers the development phase. After you have completed this tutorial, you will see instructions for starting a tutorial that covers the administration phase.

Steps in This Tutorial

- [Step 1: Start WebLogic Workshop, Open an Application, and Start WebLogic Server](#) – 5 minutes

In this step you start WebLogic Workshop, open the sample portal application, and start WebLogic Server.

- [Step 2: Add a Portal to the Application](#) – 5 minutes

In this step you Create a Portal Application, add a Portal Web Project, and add a Portal, and start WebLogic Server.

- [Step 3: Create a Simple Navigation Page Flow](#) – 10 minutes

In this step you create two simple .jpf files and their corresponding portlets.

- [Step 4: Create Portlets that Communicate](#) – 10 minutes

In this step you create a Page Flow portlet that listens to input from another Page Flow portlet.

- [Step 5: Create an EJB Control Page Flow Portlet](#) – 10 minutes

In this step you create a Page Flow portlet, add a Personalization Control, and edit properties on the Control.

Step 1: Start WebLogic Workshop, Open an Application, and Start WebLogic Server

In this step, you will start the development environment for building a portal.

The tasks in this step are:

- [Start WebLogic Workshop](#)
- [Open the Sample Portal Application](#)
- [Start WebLogic Server](#)

Start WebLogic Workshop

- **Windows** - Choose **Start**→**Programs**→**BEA WebLogic Platform 8.1**→**WebLogic Workshop**. The start script is located in the <BEA_HOME>/weblogic81/workshop directory.

Open the Sample Portal Application

The portal application you open contains all the necessary portal resources to complete the tutorial. In Java 2 Enterprise Edition (J2EE) terms, this application is an enterprise application that contains Web applications and related resources. Each Web application can contain multiple portals.

1. In the WebLogic Workshop menu, choose **File**→**Open**→**Application**.
2. In the Open **Workshop Application** dialog, select the <BEA_HOME>\weblogic81\samples\portal\portalApp\portalApp.work file and click **Open**.

The portalApp application directory tree appears in the **Application** window.

Start WebLogic Server

To develop portals and portal applications in WebLogic Workshop, WebLogic Server must be running on your development machine. For this tutorial, you will start the domain server used by the WebLogic Portal samples. The portalApp application you opened in the previous step contains all the necessary server configuration settings. To start WebLogic Server:

- In the WebLogic Workshop menu, choose **Tools**→**WebLogic Server**→**Start WebLogic Server**.

On Windows systems, you can bring up the command window from the Windows task bar to watch the startup progress. When the server starts, the WebLogic Workshop status bar shows the message "Server Running."

Step 2: Add a Portal to the Application

In this step, you will add a portal to the portal application.

Add a Portal

Create a Portal File to use as the layout framework to view your portlets. Name this **portal.portal**.

Step 3: Create a Simple Navigation Page Flow

In this step, you will create a Page Flow that provides navigation from one JSP to another. Then, instead of designating the JSP files as the content URL for the portlet, you will designate the .JPF file as the portlet's content node. The Page Flow supports JSP transitions within the portlet.

The tasks in this step are:

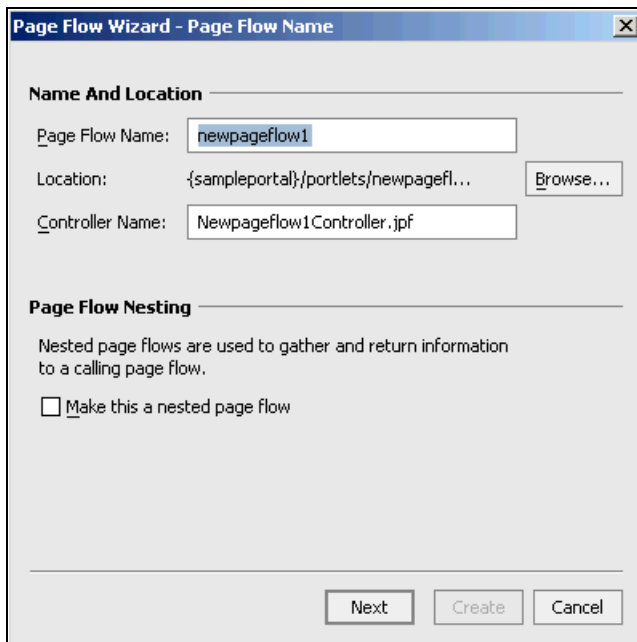
- Create a Page Flow Called simpleFlow
- Test the Page Flow in a portlet

Create a Page Flow called simpleFlow

1. Right-click a project in your web application and create a new folder; name the folder "portlets".

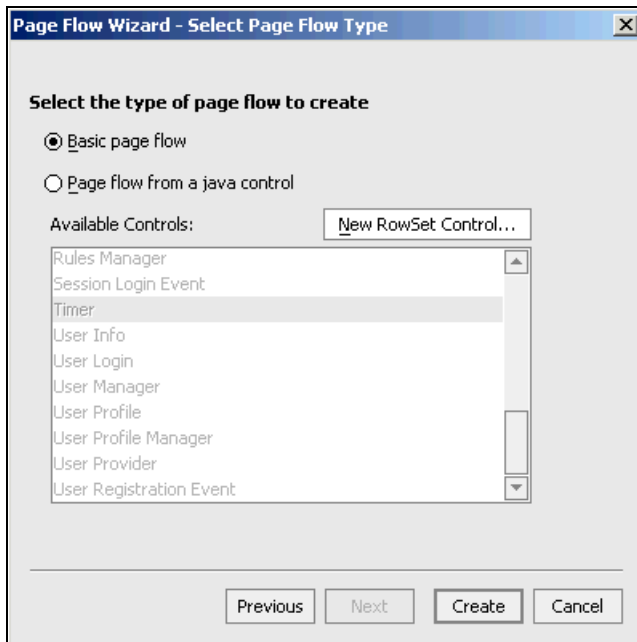
2. Right-click on the portlets folder and select **New > Page Flow**.

The Page Flow Name dialog appears. You will use this dialog to create a basic Page Flow called "simpleFlow".



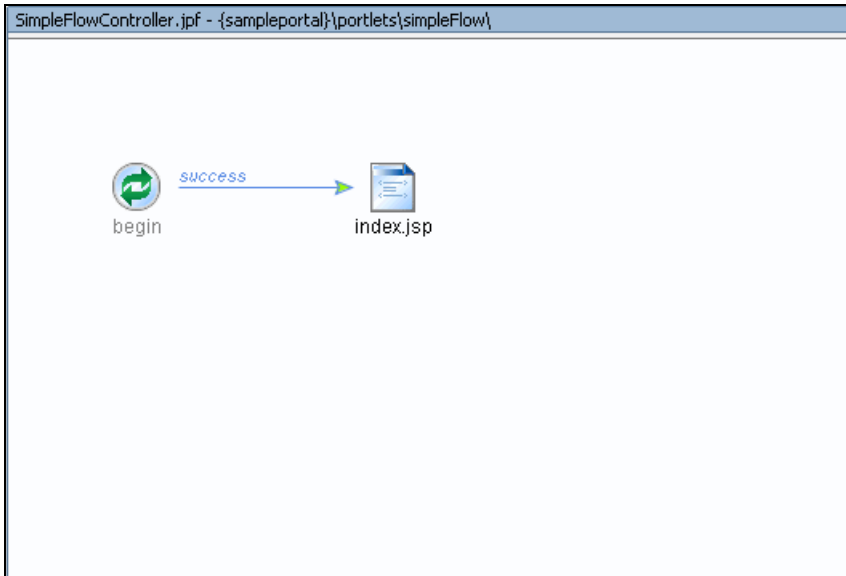
3. Name the page flow "simpleFlow" and click **Next**.

The Select Page Flow Type window appears.



4. Accept the default (**Basic page flow**) and click **Create**.

The page flow is generated and the PageFlow editor, in the Flow View, appears.

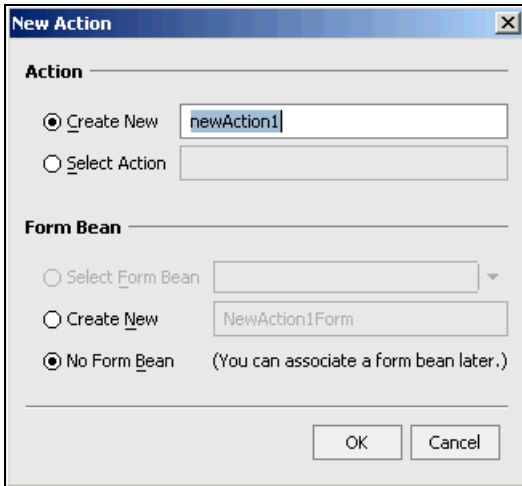


Note that the basic page flow has a **begin** action and an `index.jsp` presentation page.

5. Click **Page** in the palette, and drag a new page onto the PageFlow editor.
6. Name the page "page2.jsp".
7. Connect the two pages by doing the following:

- a. Click **Action** in the palette and drag a new action onto the PageFlow editor.

The New Action window appears.



- b. Name the action "goToPage2" and click **OK**.

An icon for the **goToPage2** action appears in the Page Flow editor.

- c. Connect **index.jsp** to **goToPage2** by placing your pointer near the **index.jsp** icon, clicking to create the connection, and dragging to **goToPage2**.
- d. Connect **goToPage2** to **page2.jsp** (as described in step c., above). A red error indicator will appear under **index.jsp**. This will be fixed later in this procedure.

8. Create another action from the palette and name it "goToPage1" on the New Action window. Click **OK**.

The **goToPage1** action appears in the Page Flow editor.

9. Using the procedure described in step 7, connect the **goToPage1** action to **index.jsp** and then connect **page2.jsp** to **goToPage1**.
10. Correct the error indicators on index.jsp by doing the following:
 - a. Double-click **index.jsp** to open the JSP editor (ensure you are in the Source View).
 - b. Change the text within the <p> </p> elements to "PageFlow Page1".

- c. Drag and drop the **goToPage2** action from the data palette onto index.jsp, at the end of "PageFlow Page1." The paragraph block will look like this:

```
<p>  
PageFlow Page1  
<netui:anchor action="goToPage2">show_goToPage2</netui:anchor>  
</p>
```

- d. Save and close index.jsp.

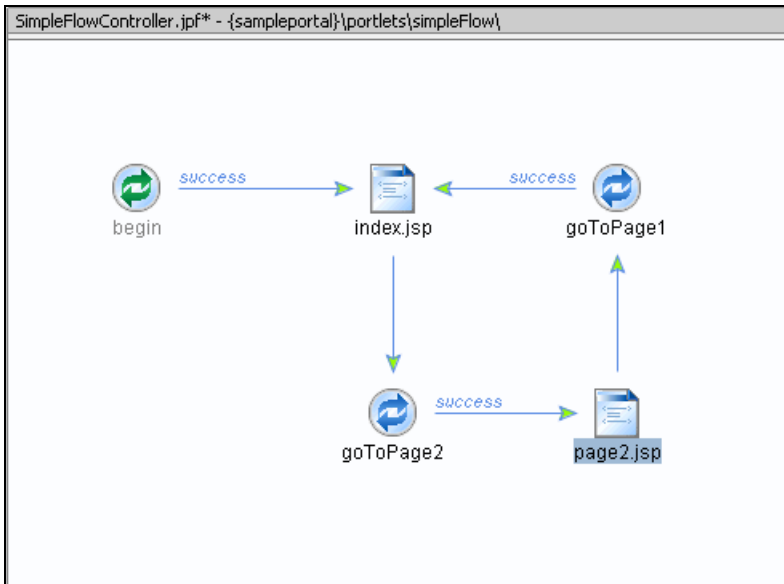
11. Correct the error indicators on page2.jsp by doing the following:

- a. Double-click **page2.jsp** to open the JSP editor (ensure you are in the Source View).
- b. Change the text within the <p> </p> elements to "PageFlow Page2".
- c. Drag and drop the **goToPage1** action from the data palette onto page2.jsp, at the end of "PageFlow Page2." The paragraph block will look like this:

```
<p>  
PageFlow Page2  
<netui:anchor action="goToPage1">show_goToPage1</netui:anchor>  
</p>
```

- d. Save and close page2.jsp.

Your page flow should now look like the following picture. Move the icons into the configuration below to make sure the relationships are correct.



12. Test the page flow in the test browser by doing the following:

- a. Click the Start button ().

The page flow will appear in the Test Browser:



- b. Click **show_gotoPage2**.
The page2.jsp appears.
- c. Click **show_gotoPage1** to return to PageFlow page 1.

Test the Page Flow in a Portlet

1. Create a portal and drag simpleFlowController.jspf into it.
The Portlet Wizard appears.
2. Accept the Portlet Wizard defaults and click finish to create the portlet.
3. Save the portal and then view it in a browser by choosing **Portal > Open Current Portal**.

JSP Examples

The JSPs should appear roughly as follows:

index.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>
  <head>
    <title>
      Web Application Page
    </title>
  </head>
  <body>
    <p>
      PageFlow page 1
      <netui:anchor action="goToPage2">show_goToPage2</netui:anchor>
    </p>
  </body>
</netui:html>
```

page2.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>
  <head>
    <title>
      Web Application Page
    </title>
  </head>
  <body>
    <p>
      PageFlow Page 2
      <netui:anchor action="goToPage1">show_goToPage1</netui:anchor>
    </p>
  </body>
</netui:html>
```

Step 4: Create Portlets that Communicate

In this step, you will use Page Flows to create one portlet that listens to another portlet. This particular example shows two ways of passing information from one portlet to the next: using a Form or using the Request Parameter. They both use the Page Flow as the means of connecting

two portlets. By configuring the portlets to listen to one another, forms and requests instantiated by a JSP in one Page Flow can be sent to both Page Flows. This allows the listening portlet to update itself.

The tasks in this step are:

- [Create two Page Flows](#)
- [Edit Page Flows](#)
- [Create and Place Portlets in the Portal](#)
- [Set Properties on Portlets](#)

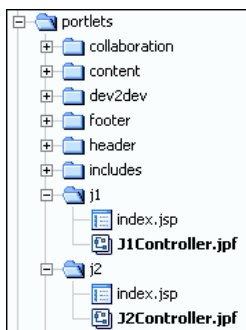
Note: Case sensitivity in naming directories and files in this step is important, especially if you are copying and pasting the code samples from this step.

Create two Page Flows

In this task, two Page Flows are created within the portlets directory; one called **j1**, the other called **j2**.

1. Right-click on the portlets directory, and select **Create New Page Flow**.
2. Name the Page Flow **j1**, and select **Basic Page Flow** (with no Java Controls.)
3. Repeat this step, creating a Page Flow called **j2**.

Note: WebLogic Workshop automatically creates the Page Flow controller class, prepending the name of the Page Flow to the Controller.jspf filename.



Edit Page Flows

This example requires modifying the Page Flows so that they can be notified by the listening portlet. The **j1** Page Flow will receive form data from its **simpleForm.jsp**. The **j2** Page Flow (in a listening portlet) will also receive notifications, and store the data, so its portlet can retrieve and display it.

Note: Use Flow View to set up actions, then touch up the code using the Source View.

Edit j1 Page Flow

In the following steps, you will add a reference to a JSP that does not yet exist. You will create this JSP later in this topic.

1. Open the **j1Controller.jspf** and make sure the package declaration is correct:

```
package portlets.j1;
```

2. Edit the begin method, adding a forward named simpleForm:

```
/**
 * @jpf:action
 * @jpf:forward name="simpleForm" path="simpleForm.jsp"
 */
protected Forward begin()
{
    return new Forward( "simpleForm" );
}
```

3. Add the following three actions to the Page Flow, in front of the Page Flow's closing bracket "}":

```
/**
 * @jpf:action
 * @jpf:forward name="simpleForm" path="simpleForm.jsp"
 */
protected Forward passString1( Form form )
{
    String passedText = form.getText();
    return new Forward( "simpleForm" );
}

/**
 * @jpf:action
 * @jpf:forward name="simpleForm" path="simpleForm.jsp"
 */
protected Forward passString2()
```

```

    {
        return new Forward( "simpleForm" );
    }

/**
 * @jpf:action
 * @jpf:forward name="simpleForm" path="simpleForm.jsp"
 */
protected Forward passString3()
{
    return new Forward( "simpleForm" );
}

```

4. Make sure the following Form inner class appears at the end of the Page Flow file, in front of the Page Flow's closing bracket "}

```

public static class Form extends FormData
{
    private String text;
    public void setText( String text )
    {
        this.text = text;
    }
    public String getText()
    {
        return this.text;
    }
}

```

5. Press Alt+Enter to add an import statement for the com.bea.wlw.netui.pageflow.FormData class.

Edit j2 Page Flow

In the following steps, you will add a reference to a JSP that does not yet exist. You will create this JSP later in this topic.

1. Open the **j2Controller.jspf** and make sure the package declaration is correct:

```
package portlets.j2;
```

2. This Page Flow receives data from the **j1** portlet and stores it so it can be displayed in the **j2** portlet. This directs the j2 portlet to listen to the **j1** portlet. Add a variable declaration to the beginning of the class to hold the text from the j1 Page Flow:

```
public String thePassedText = "";
```

3. Edit the begin method, adding a forward named **listening**, which points to the **listening.jsp** we'll create later. The listening.jsp will display in the j2 portlet the data received from the j2 Page Flow.

```
/**
 * @jpf:action
 * @jpf:forward name="listening" path="listening.jsp"
 */
public Forward begin()
{
    return new Forward( "listening" );
}
```

4. The first action we'll now add to the Page Flow has the same signature as the **j1** passString1 method. When the **j2** portlet is listening to **j1**, both the **j1.passString1** method and the **j2.passString1** methods are called.

```
/**
 * @jpf:action
 * @jpf:forward name="listening" path="listening.jsp"
 */
public Forward passString1(portlets.j1.J1Controller.Form form)
{
    thePassedText = form.getText();
    return new Forward( "listening" );
}
```

5. If prompted by the tooltip, press Alt+Enter to import javax.servlet.http.HttpServletRequest.
6. To illustrate other ways to pass strings between Page Flows, add passString2 and passString3 to the Page Flow for portlet **j2**.

- The [passString2](#) method uses a request to send the data.
- The [passString3](#) method uses the same portlet communication approach as passString2; the only difference is that the **j2** Page Flow sends data to its JSP portlet using an attribute.

```
/**
 * @jpf:action
 * @jpf:forward name="listening" path="listening.jsp"
 */
public Forward passString2()
{
    thePassedText = getRequest().getParameter("string2");
    return new Forward( "listening" );
}
```

```

/**
 * @jpf:action
 * @jpf:forward name="listening" path="listening.jsp"
 */
public Forward passString3()
{
    HttpServletRequest request = getRequest();
    String attribValue = request.getParameter("string3");
    request.setAttribute("string3", attribValue);
    return new Forward( "listening" );
}

```

Create Additional JSPs

In this step, we'll create the "From" JSP for the **j1** portlet, and the "To" JSP used by the **j2** portlet.

Create listening.jsp

1. In the Application palette, right-click on the **portlets/j2** folder and select **New JSP file**.
2. Name the file **listening.jsp**.
3. Open the source view, and insert the following code:

```

<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<%@ taglib uri="netui-tags-html.tld" prefix="netui" %>
<h3>"To" JSP</h3>
Get text from jpf form = <b><netui:label
value="{pageFlow.thePassedText}" /></b>
<br/>
<br/>
Using {request.string3} databinding = <b> <netui:label
value="{request.string3}" />
</b> <br/>
<% String attribName = "string3"; %>
Using request.getAttribute() = <%=request.getAttribute(attribName)%>
<br/>
<br/>

```

4. Save **listening.jsp**.

Create simpleForm.jsp

1. In the Application palette, right-click on the **portlets/j1** folder and select **New JSP file**.

2. Name this new JSP file **simpleForm.jsp**.
3. Switch to Source View.
4. Delete the "New Web Application Page" text.
5. From the NetUI Palette, drag a TextBox object into the paragraph, then a Button object right below it. In the New Button dialog box that appears, enter a Value of **Submit**.

Manually enter opening and closing `<netui:form>` tags around the text box and button (selecting an action of "none"). Initially, the source for this portion of the jsp will look something like this:

```
<netui:form action="none">
<netui:textBox></netui:textBox>
<netui:button>Submit</netui:button>
</netui:form>
```

6. Edit the source of the fragment you created: Add the `passString1` action to the form, and designate a `dataSource` attribute for the `textBox`:

```
<netui:form action="passString1">
<netui:textBox dataSource="text"/>
<netui:button>Submit</netui:button>
</netui:form>
```

Note: You can click in each tag and type the action and `dataSource` attribute values in the Property Editor window.

7. From the NetUI Palette, drag an anchor element just after the closing `</p>` (but before the closing `</body>` tag). The Form Wizard appears. Invoke the `passString2` action, labeling it "PassString2".
8. From the NetUI Palette, drag a parameter element just in front of the closing `</netui:anchor>` tag. Change the name to "string2" and the value to "STRING 2". The tags should look like this:

```
<netui:anchor action="passString2">Pass String 2
<netui:parameter name="string2" value="STRING 2"/>
</netui:anchor>
```

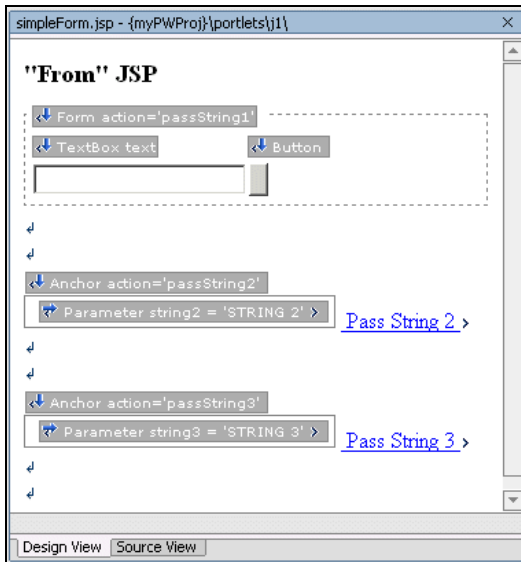
9. Copy the previous code block and paste it below the previous code block and in front of the closing `</body>` tag. Change the values from 2 to 3 to look like the following code:

```
<netui:anchor action="passString3">Pass String 3
<netui:parameter name="string3" value="STRING 3"/>
</netui:anchor>
```


10. At the top of the JSP content area, add the following HTML:

```
<h3>"From" JSP</h3>
```

11. The Design View of **simpleForm.jsp** should now look something like this:



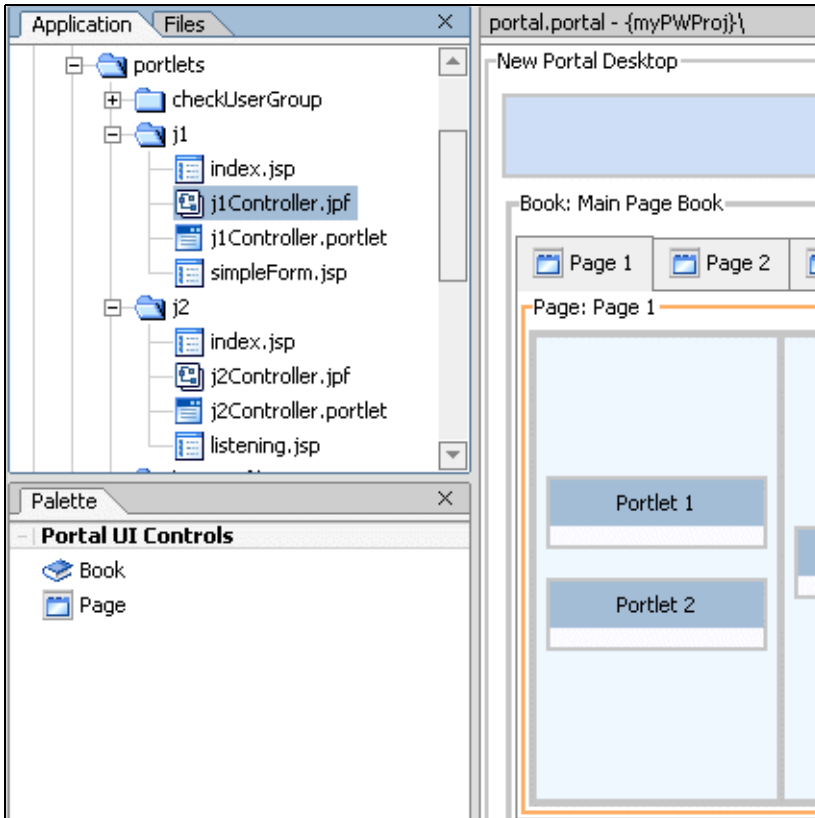
Note: As you edit the page flow, you can verify the navigation by opening a viewer that will preview the pages without the portal. To do this, click on the **Start** arrow from the WebLogic Workshop toolbar, or press **CTRL+F5**.

Create and Place Portlets in the Portal

To view the portlets, they need to be placed in a portal.

1. Open the portal created in Step 1 in Design View, select a placeholder, and drag the Page Flows into placeholders on a page, using the Portlet Wizard to create new portlets from each Page Flow.
2. Use the Property designer to edit the Title attribute for each portlet.

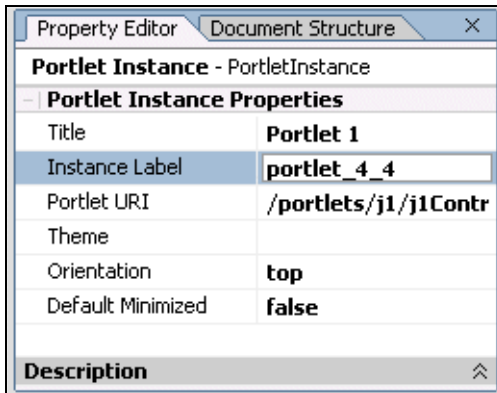
3. From the Portal designer, the portal should now look something like this:



Set Properties on Portlets

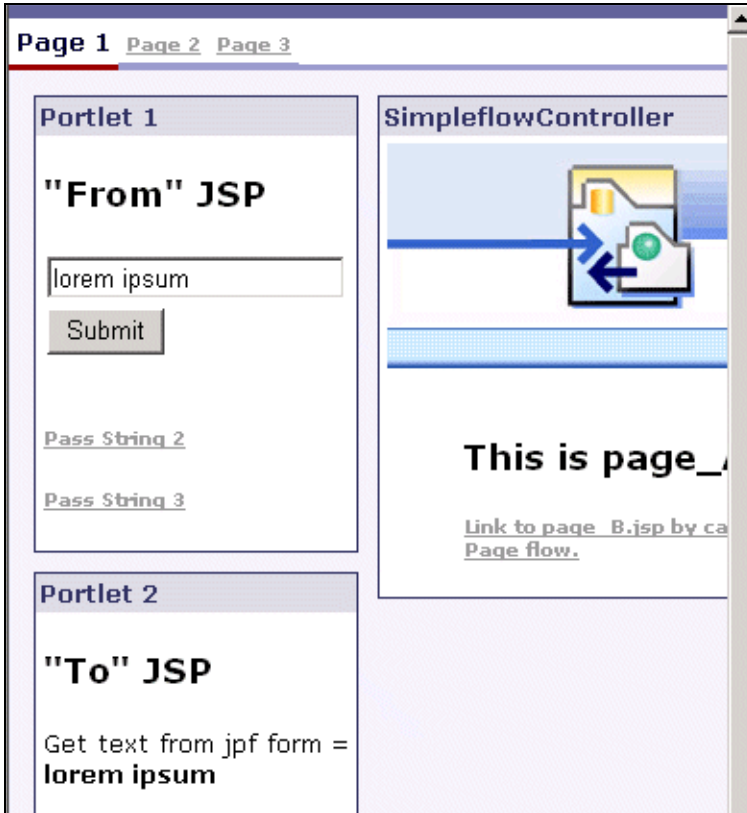
When a portlet is placed inside a portal, it is assigned an **instanceLabel** which the framework uses to keep track of individual portlet placement. In order to make this sample work, the `listenTo` attribute on portlet **j2** needs to be set to the `instanceLabel` of the **j1** portlet on your portal.

1. Open the portal, select the **j1** portlet and verify its instanceLabel. In this example, it is **portlet_4_4**.



2. Now open the **j2** portlet by double-clicking on it within the portal. The listenTo attribute for this portlet needs to be set to the instanceLabel for the j1 portlet in your portal.
3. Save all files and start the server.
4. Preview the portal by navigating to **http://<host>:<port>YourWebapp/portal.portal**.

5. The resulting portal should look something like this:



Step 5: Create an EJB Control Page Flow Portlet

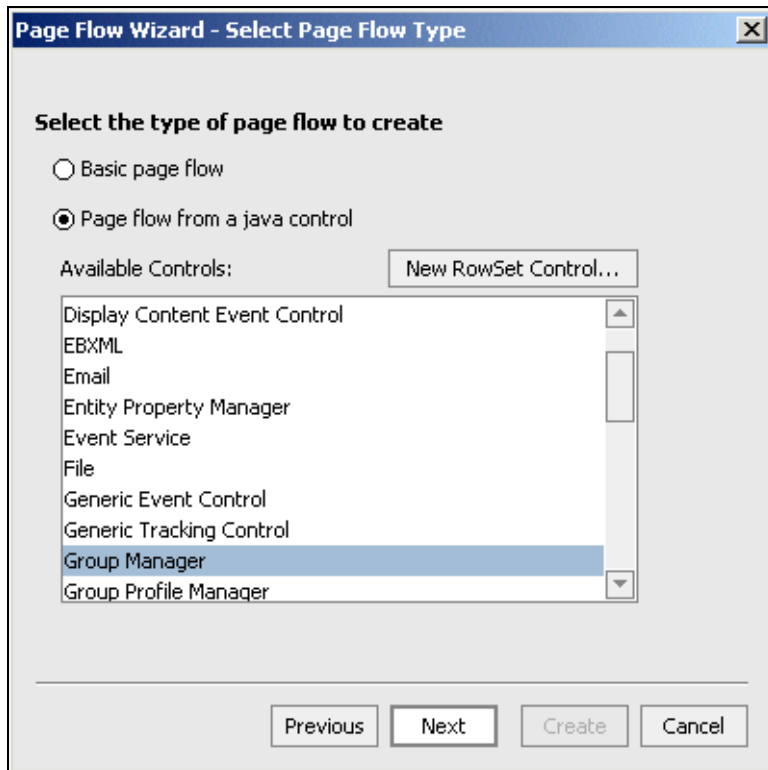
In this step you create a Page Flow portlet, add a Personalization Control, and edit properties on the Control.

The tasks in this step are:

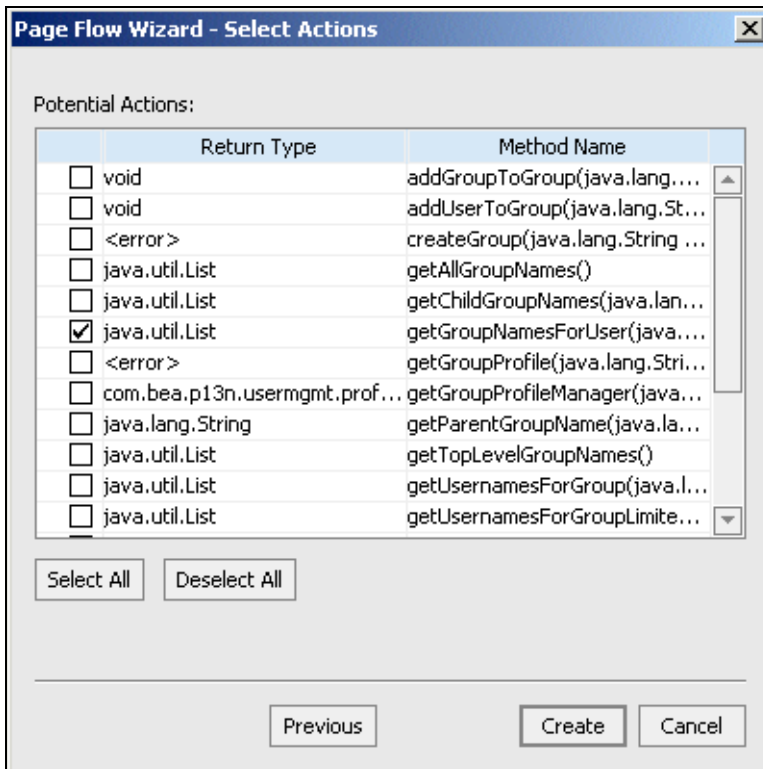
- Create an EJB Control Page Flow
- Place the Page Flow into a Portlet, Place Portlet in a Portal

Create a New Page Flow

1. From within the Project tab in your Web application, right-click on the portlets directory and create a new Page Flow. Name the Page Flow **checkuserGroup**.
2. In the Page Flow Wizard, select **Page flow from a java control**.
3. Select the **GroupManager** control. Click **Next**.



- From the list of Potential Actions, select **getGroupNamesForUser**. Click **Create**.

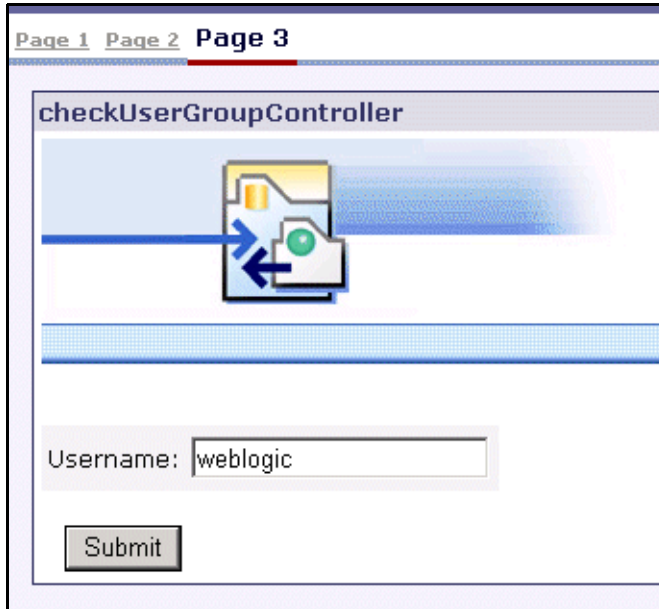


- The wizard creates an index.jsp page with a link to the getGroupNamesForUser action that invokes the control.

Place the Page Flow into a Portlet, Place Portlet in a Portal

- With the Portal open in Design View, drag the **checkUserGroupController.jspf** into a placeholder in the portal.
- The Portlet Wizard presents a prompt offering to create a portlet from this resource. Click **Yes**.
- The Portlet Details screen appears. Click **Finish**.
- Preview the portlet by navigating to **http://<host>:<port>YourWebapp/portal.portal**.

5. Click on the `getGroupNamesForUser` link, enter a username and click **Submit**.



The screenshot shows a web application interface. At the top, there is a navigation bar with three tabs: "Page 1", "Page 2", and "Page 3", with "Page 3" being the active tab. Below the navigation bar is a header area with the text "checkUserGroupController". The main content area features a large blue horizontal bar with a document icon containing a green circle and a blue arrow pointing left. Below this bar is a text input field labeled "Username:" with the value "weblogic" entered. At the bottom of the form is a "Submit" button.

6. In this example, user **weblogic** is a member of the groups **Administrators** and **PortalSystemAdministrators**.

The screenshot shows a web application interface with a breadcrumb trail at the top: [Page 1](#) [Page 2](#) **Page 3**. Below the breadcrumb is a header area with the text **checkUserGroupController**. A central graphic depicts a folder icon with a blue arrow pointing into it and another blue arrow pointing out of it, set against a blue gradient background. Below this graphic is a section titled **Page Flow: Check Groups for User**. Underneath the title is a grey box containing the text **Input Forms For Actions With Parameters** and the method name `getGroupNamesForUser`. Below that is another grey box labeled **Results Area**, which contains the text `[Administrators, PortalSystemAdministrators]`.