

Oracle® Fusion Middleware

Tutorial for Oracle WebCenter Developers

11g Release 1 (11.1.1)

E10273-01

May 2009

Oracle Fusion Middleware Tutorial for Oracle WebCenter Developers, 11g Release 1 (11.1.1)

E10273-01

Copyright © 2007, 2009, Oracle and/or its affiliates. All rights reserved.

Primary Author: Vanessa Wang

Contributor: Peter Moskovits, Robin Fisher

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this software or related documentation is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, duplication, disclosure, modification, and adaptation shall be subject to the restrictions and license terms set forth in the applicable Government contract, and, to the extent applicable by the terms of the Government contract, the additional rights set forth in FAR 52.227-19, Commercial Computer Software License (December 2007). Oracle USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

This software is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications which may create a risk of personal injury. If you use this software in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure the safe use of this software. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software in dangerous applications.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

This software and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

Contents

Preface	v
Audience	v
Documentation Accessibility	v
Related Documents	vi
Conventions	vi
1 Introduction to WebCenter Framework and the Tutorial	
What is WebCenter Framework?	1-1
What Will I Create?	1-2
2 Preparing for the Tutorial	
Introduction	2-1
Step 1: Obtain the Software	2-1
Step 2: Download the Sample Tutorial Files	2-2
Step 3: Add the Tutorial Sample Schema to Your Database	2-3
Step 4: Install the WebCenter Schema	2-5
3 Creating a WebCenter Application with a Customizable Page	
Introduction	3-1
Step 1: Create a Custom WebCenter Application	3-2
Step 2: Add the Images Files to the Application	3-6
Step 3: Create a Page	3-8
Step 4: Add Layout Components to the Page	3-11
Step 5: Add Oracle Composer to the Page to Enable Customization	3-25
Step 6: Customize the Page at Runtime Using Oracle Composer	3-31
4 Adding WebCenter Web 2.0 Services to Your Application	
Introduction	4-2
Step 1: Add the Search Toolbar Task Flow to the Application	4-2
Step 2: Create a Connection for the Documents Service	4-6
Step 3: Add the Document Library Task Flow to Your Application	4-10
Step 4: Browse Documents at Runtime	4-15
Step 5: Create a Database Connection to the WebCenter Schema for the Tags Service	4-18
Step 6: Add the Tags Service to Your Application	4-19

Step 7: Use, Add, and Search Tags in Your Application at Runtime	4-21
--	------

5 Building Portlets and Wiring Them in Your Application

Introduction.....	5-2
Step 1: Create a Standards-Based Java (JSR 168) Portlet.....	5-3
Step 2: Create the JavaBeans to Store the Standards-Based Portlet Information	5-15
Step 3: Create the Business Logic for the Standards-Based Portlet.....	5-23
Step 4: Test and Deploy the Standards-Based Portlet	5-31
Step 5: Register the Standards-Based Portlet with Your Application	5-36
Step 6: Test the Standards-Based Portlet in Your Application.....	5-38
Step 7: Register the Preconfigured Portlet Producer	5-40
Step 8: Add an OmniPortlet to Your Page.....	5-43
Step 9: Define OmniPortlet at Runtime.....	5-46
Step 10: Wire the Standards-Based Portlet and OmniPortlet Together	5-53
Step 11: Test the Interaction Between the Portlets	5-56

6 Conclusion

Summary	6-1
Moving On.....	6-2

Index

Preface

This tutorial introduces you to Oracle WebCenter Framework, a key component of Oracle WebCenter Suite that enables you to build your own custom WebCenter applications. As you work through this tutorial, you'll become familiar with Oracle JDeveloper and the components that have been added to support the new Oracle WebCenter Framework functionality. When you're ready to begin building your own application, you can move on to the *Oracle Fusion Middleware Developer's Guide for Oracle WebCenter* for assistance.

If you are looking for a pre-built sample application, you can check out the Fusion Order Demo for WebCenter, located on the Oracle WebCenter Suite page on the Oracle Technology Network (OTN) at <http://webcenter.oracle.com>.

Note: For the portable document format (PDF) version of this manual, when a URL breaks onto two lines, the full URL data is not sent to the browser when you click it. To get to the correct target of any URL included in the PDF, copy and paste the URL into your browser's address field. In the HTML version of this manual, you can click a link to directly display its target in your browser.

Audience

This document is intended for users wishing to familiarize themselves with Oracle WebCenter Framework and learn how to develop custom WebCenter applications.

This tutorial does not assume any prior knowledge of Oracle JDeveloper or Oracle WebCenter Suite. It does, however, assume that you are already somewhat familiar with the following:

- Oracle Application Development Framework (Oracle ADF)
- Oracle ADF Faces
- Java

The tutorial is intended for the developer who wants to build a custom WebCenter application, or the application developer who wants to use Oracle ADF to build customization capabilities into their application.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible to all users, including users that are disabled. To that end, our documentation includes features that make information available to users of assistive

technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at <http://www.oracle.com/accessibility/>.

Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

Deaf/Hard of Hearing Access to Oracle Support Services

To reach Oracle Support Services, use a telecommunications relay service (TRS) to call Oracle Support at 1.800.223.1711. An Oracle Support Services engineer will handle technical issues and provide customer support according to the Oracle service request process. Information about TRS is available at <http://www.fcc.gov/cgb/consumerfacts/trs.html>, and a list of phone numbers is available at <http://www.fcc.gov/cgb/dro/trsphonebk.html>.

Related Documents

For more information on Oracle WebCenter Framework, see the following documents, which are available on the Oracle WebCenter Suite Documentation page on the Oracle Technology Network (OTN) at

<http://www.oracle.com/technology/products/webcenter/documentation.html>:

- Oracle Fusion Middleware Developer's Guide for Oracle WebCenter, which explains how to use Oracle JDeveloper and Oracle WebCenter Framework to *develop* custom WebCenter applications
- Oracle Fusion Middleware User's Guide for Oracle WebCenter, which explains how to *use* custom WebCenter applications at runtime (in a browser)

For more information on Application Development Framework, see the Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
<i>italic</i>	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.

Convention	Meaning
<code>monospace</code>	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Introduction to WebCenter Framework and the Tutorial

Welcome to Oracle WebCenter Framework! This chapter introduces you to key Oracle WebCenter Framework concepts, then explains what you will create during the steps in this tutorial. The lessons are designed to familiarize you with different aspects of WebCenter Framework functionality, and to demonstrate enough about each feature so that you can create your own custom WebCenter applications.

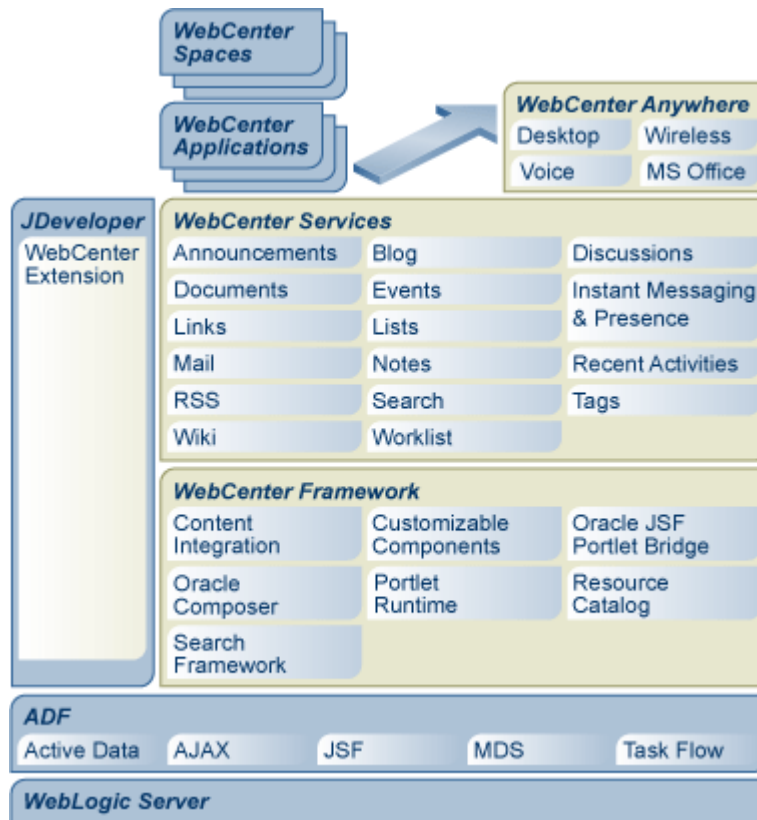
If you need additional information about a feature, you can always refer to the Oracle Fusion Middleware Developer's Guide for Oracle WebCenter and the Oracle Fusion Middleware User's Guide for Oracle WebCenter.

What is WebCenter Framework?

Oracle WebCenter Framework is a declarative JavaServer Faces (JSF)-based framework that enables embedding of AJAX-based components, services, portlets, and content into context-rich customizable applications. Leveraging a revolutionary way of layered customizations, these applications and portals store user changes in Oracle Metadata Services that is used across all of Oracle Fusion Middleware and is the foundation for Fusion Applications. It insulates users and developers from patching and upgrades to speed new capabilities to make businesses more agile, and is delivered as an extension to Oracle JDeveloper to provide an integrated development environment for composite Java EE applications, business processes, BI applications, and enterprise portals.

[Figure 1-1](#) provides an overview of the Oracle WebCenter architecture, showing the major components that make up the product and the features and services offered.

Figure 1–1 Overview of the Oracle WebCenter Architecture



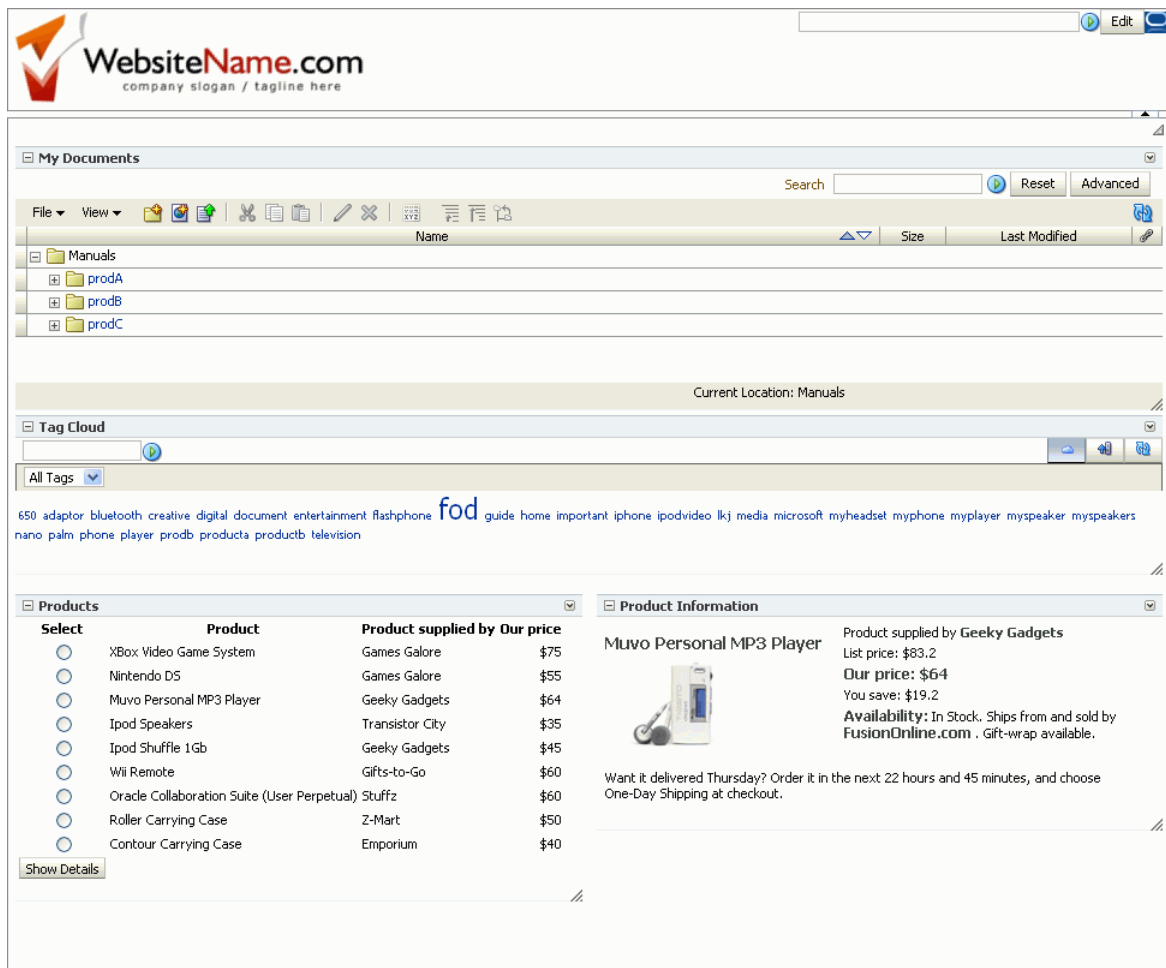
For more information about Oracle WebCenter, refer to Chapter 1, “Understanding Oracle WebCenter” in the Oracle Fusion Middleware Developer’s Guide for Oracle WebCenter.

What Will I Create?

In this tutorial, you will use WebCenter Framework to build a custom WebCenter application that is customizable at runtime, empowering you and your end users to edit application pages according to personal requirements and directly leveraging Oracle Metadata Services. You will also use WebCenter Services to integrate content from a content repository and display it in a user-friendly interface, and enable users to “tag” and search the content. You will build and consume two types of portlets: a rich, standards-based portlet and an out-of-the-box PDK-Java portlet that you define using a wizard. Finally, you will enable interaction between the two portlets, so that user actions on one portlet drives the content that displays in the second portlet.

Figure 1–2 shows the custom WebCenter application you will create in this tutorial.

Figure 1–2 Final Tutorial Application



This tutorial is designed for the chapters to be completed in the same sequence as they are presented. Due to dependencies, completing them in a different order may result in missing resources or even errors.

The path through this tutorial is as follows:

- [Chapter 2, "Preparing for the Tutorial"](#) tells you what you must do before you can complete the steps in this tutorial, including installing the resource files for the sample application you will build. Be sure to complete all the steps described in this chapter.
- [Chapter 3, "Creating a WebCenter Application with a Customizable Page"](#) introduces you to creating a custom WebCenter application, creating a JSF page, and enabling runtime customization with Oracle Composer. You will also use Oracle Composer to customize your application at runtime.
- [Chapter 4, "Adding WebCenter Web 2.0 Services to Your Application"](#) shows you how to add various services to your application that enable your users to access content on a file system by using a document library, search for content across the application, and add tagging and a tag cloud to your application. You will also learn how to use each of these services at runtime.
- [Chapter 5, "Building Portlets and Wiring Them in Your Application"](#) tells you how to create two types of portlets: an OmniPortlet and a simple standards-based Java

(JSR 168) portlet. You will also enhance the JSR 168 portlet to embrace more sophisticated logic. You will then enable these two portlets to communicate with each other, so that when you select an option in the first (JSR 168) portlet, the content of the second portlet (OmniPortlet) updates based on that selection.

Preparing for the Tutorial

This chapter tells you how to obtain the sample files and install the tutorial and Oracle WebCenter database schemas required for completing this tutorial. These files and database schemas are necessary for building the complete sample application. You must have administrator's access to the database where you'll install the database schemas.

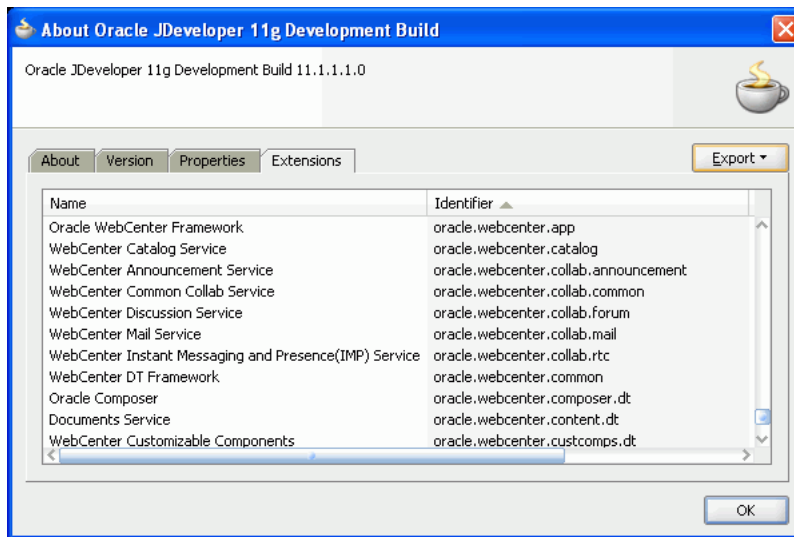
Introduction

We will set up the environment for the tutorial by following these steps:

- [Step 1: Obtain the Software](#)
- [Step 2: Download the Sample Tutorial Files](#)
- [Step 3: Add the Tutorial Sample Schema to Your Database](#)
- [Step 4: Install the WebCenter Schema](#)

Step 1: Obtain the Software

Ensure that you have installed Oracle JDeveloper 11g Release 1 (11.1.1) and the Oracle WebCenter extension (11.1.1). If you are not sure whether you have the WebCenter extension, you can verify this by opening Oracle JDeveloper, then choosing **Help > About** from the menu, then click the **Extensions** tab. On the Extensions list, sort by **Identifier** to locate the `oracle.webcenter.*` components. [Figure 2-1](#) shows the Oracle WebCenter components listed in JDeveloper.

Figure 2–1 Oracle WebCenter Framework in Oracle JDeveloper

If you do not see these components, you must install the WebCenter extension.

To install the WebCenter extension to Oracle JDeveloper using the Update Center:

1. Launch Oracle JDeveloper.
2. If the Select Default Roles dialog box displays, select **Default Role** to enable all technologies, and click **OK**.
3. If a dialog box displays asking if you want to migrate settings from an earlier version, click **No**.
4. In Oracle JDeveloper, choose **Help > Check for Updates**.
5. On the Welcome page, click **Next**.
6. Select **Search Update Centers**, then click **Next**.
7. On the Updates page, search for the WebCenter extension, select it, then click **Finish**.
8. When prompted, restart JDeveloper.

For more information on obtaining and installing Oracle WebCenter Framework, see the Oracle WebCenter page on OTN (<http://webcenter.oracle.com>).

Step 2: Download the Sample Tutorial Files

At various points throughout this tutorial, you'll be asked to include certain content and images in your application. This material is contained in a zip file, which you can download by following these instructions:

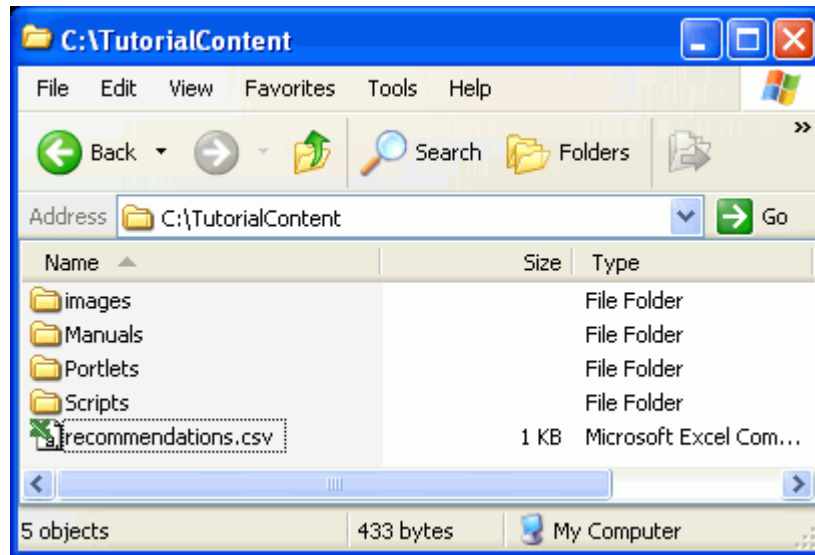
To download the sample tutorial files:

1. Open a browser, and enter the following in the Address field:


```
http://www.oracle.com/technology/products/webcenter/files/webcenter_tutorial11gr1.zip
```
2. Open the ZIP file (webcentertutorial11gR1.zip).
3. Unzip the file to a local drive, such as C.

Figure 2–2 shows the file unzipped to: C:\TutorialContent.

Figure 2–2 Sample Content ZIP File Unzipped



Step 3: Add the Tutorial Sample Schema to Your Database

Some examples we will use in this tutorial will access data using SQL. You must add the schema to your database to complete these lessons. However, if you do not have access to a database, you can still complete many of the other lessons in this tutorial.

You can either install the tutorial schema using SQL*Plus or by using Oracle JDeveloper. This section shows you how to create the database connection for the database where you will install the tutorial schema, then add the schema to the database, all within JDeveloper.

To complete the steps in this section, you will need the connection information (such as the location and port number) for your database containing the schema. Take note of this information for use later in the tutorial. Also, if the database is not on your local computer, you must modify the build script (`buildFromJDev.sql` or `build.sql`, depending on whether you are using JDeveloper or SQL*Plus to install the script) to specify the TNS alias when it reconnects.

Note: If you see an error that says:

```
DROP USER FOD CASCADE
```

```
*
```

```
ERROR at line 1:
```

```
ORA-01918: user 'FOD' does not exist,
```

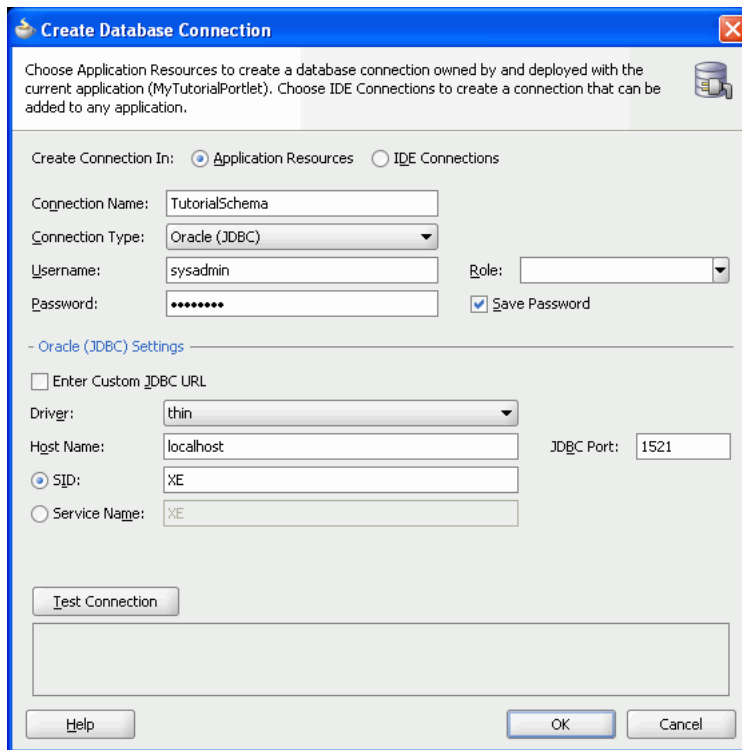
```
you can ignore this message, as it means that the schema does not yet exist.
```

To add the sample schema to your database:

1. In Oracle JDeveloper, choose **Tools > SQL Worksheet**, then click the green + sign to create a new connection.

2. In the Create Database Connection dialog box, enter connection information for the system administrator of your database (Figure 2–3):
 - **Connection Name:** TutorialSchema
 - **Connection Type:** Oracle (JDBC)
 - **User name:** <your system administrator user ID>
 - **Password:** <your system administrator password>
 - **Role:** Choose a role from the Role list (SYSDBA or SYSOPER)
 - **Host:** <host name of your database> (for example, localhost)
 - **JDBC Port:** <port> (for example, 1521)
 - **SID:** <system identifier for the database with the same JDBC port> (for example, ORCL)

Figure 2–3 Database Connection for the Tutorial Schema



3. Click **OK** to close the Create Database Connection dialog box, then click **OK** again to close the Select Connection dialog box.
4. Choose **Tools > SQL Worksheet**, then select the newly created connection.
5. Click **OK**.
6. In the SQL Worksheet panel, create the schema by enter the following command:

```
create user fod identified by fusion;
```
7. Click the **Execute Statement** icon at the top of the panel.
8. In the SQL Worksheet panel, enter the following command:

```
grant connect, resource to fod identified by fusion;
```


In doing so, you enable the credentials in the script we've provided to access the schema in your database.

- Click the **Execute Statement** icon (the green arrow) at the top of the panel.

Figure 2-4 *Execute Statement Icon*



- Choose **Tools > SQL Worksheet** again.
- In the Select Connection dialog box, click the pencil icon to edit the connection.
- Modify the connection to use the new credentials. Change the **Username** to `fod` and the **Password** to `fusion`, then click **OK** to close the Edit Database Connection dialog box.
- Click **OK** to close the Select Connection dialog box.
- Close the **SQL Worksheet** panel.
- Start a new SQL Worksheet using the new connection. Choose **Tools > SQL Worksheet** again.
- Create the schema objects by executing the `buildFromJDev.sql` script that's located in the Scripts folder (`c:\TutorialContent\Scripts`):

```
@@<path/>buildFromJDev.sql
```

You can ignore the warnings in the Log window:

```
WARNING:
java.io.PipedInputStream.checkStateForReceive(PipedInputStream.java:244)
java.io.IOException: Pipe closed
```

Note: You can also manually install the schema using SQL*Plus by using the script we've provided, `c:\TutorialContent\Scripts\build.sql`.

Step 4: Install the WebCenter Schema

To use the Tags service, you must have the WebCenter schema installed in your database. You can do this by using the built-in SQL Worksheet utility that you used in the previous step.

To install the WebCenter schema:

- From the **Tools** menu, select **SQL Worksheet**.
- In the Select Connection dialog box, click the pencil icon to edit the connection.
- Modify the connection to use an administrator username and password, such as `system` or `sys`, then click **OK**.
- Click **OK** to close the Select Connection dialog box.
- Choose **Tools > SQL Worksheet**.
- Enter the following SQL statement in the SQL Worksheet panel:

```
@@JDEV_HOME/jdeveloper/jdev/extensions/oracle.webcenter.install/sql/wc_
```

`schema.sql`

7. Click the **Execute Statement** icon, or press F9, to run the script.
8. At the prompt, enter `webcenter` as the name for the schema and a password for the schema, such as `welcome1`. The name of the schema must be `webcenter`.
9. If prompted for the Default Tablespace and Temporary Tablespace, accept the defaults (`users` and `temp`).

Now that you've set up the files and the database for your environment, you're ready to begin!

Creating a WebCenter Application with a Customizable Page

In this lesson, you will create a basic custom WebCenter application, then create a page within the application where you will later add services, content, and portlets. You will also add layout components and Oracle Composer to the page, so that you (and your users) can customize the page at runtime. At the end of the lesson, we will experiment with customizing our page at runtime using Oracle Composer.

Figure 3–1 shows how your page will look at the end of this lesson.

Figure 3–1 *MyPage.jspx at the End of this Lesson*



Introduction

This lesson contains the following steps:

- [Step 1: Create a Custom WebCenter Application](#)
- [Step 2: Add the Images Files to the Application](#)
- [Step 3: Create a Page](#)
- [Step 4: Add Layout Components to the Page](#)
- [Step 5: Add Oracle Composer to the Page to Enable Customization](#)
- [Step 6: Customize the Page at Runtime Using Oracle Composer](#)

Before you begin the steps in this lesson, ensure you have followed the steps up to this point in the tutorial.

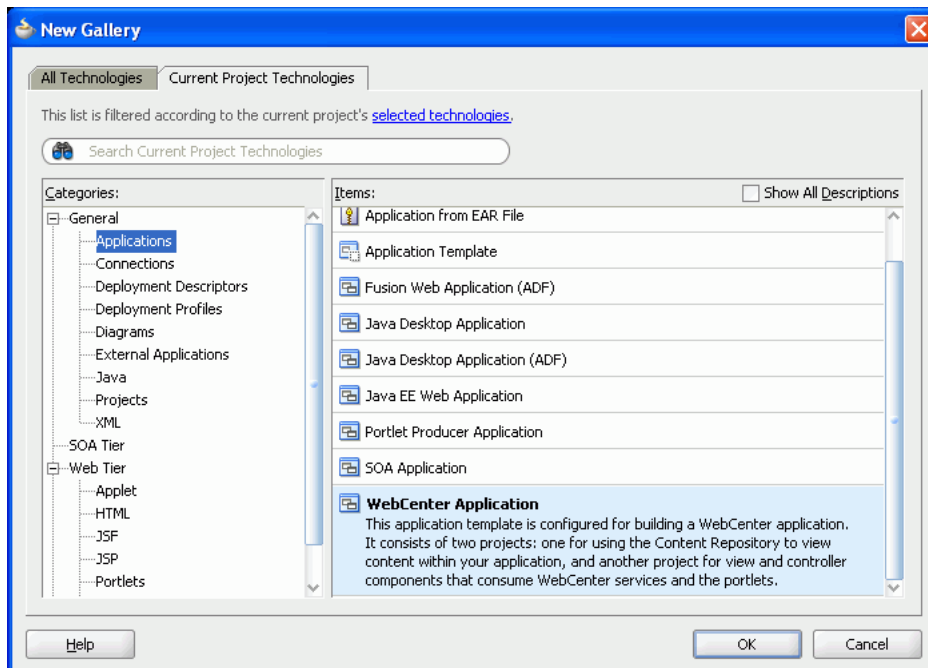
Step 1: Create a Custom WebCenter Application

Let's begin by building a simple custom WebCenter application. WebCenter Framework includes a template that you can use to quickly get started.

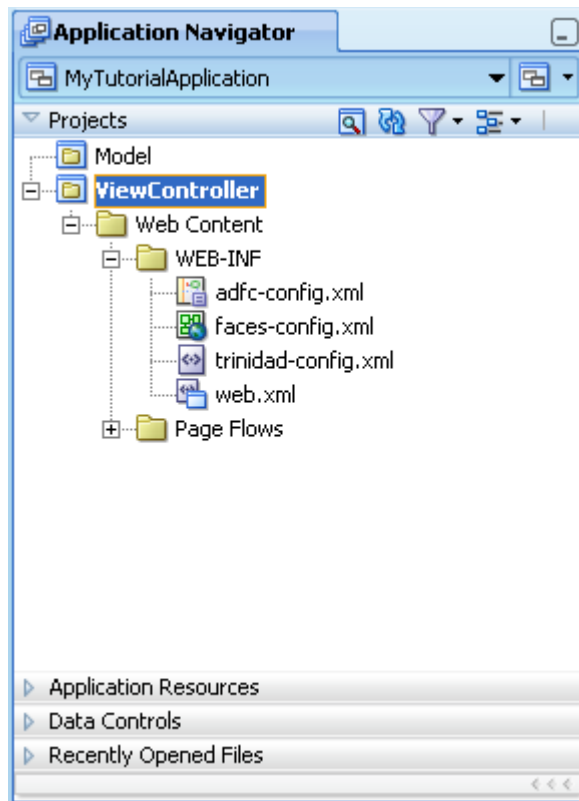
To create a custom WebCenter application:

1. In Oracle JDeveloper, choose **File > New** from the menu.
2. In the New Gallery, on the Current Project Technologies tab, you should see the General category highlighted. Under General, click **Applications**.
3. In the Items list, scroll down and select **WebCenter Application**, then click **OK** (Figure 3–2).

Figure 3–2 Create New WebCenter Application

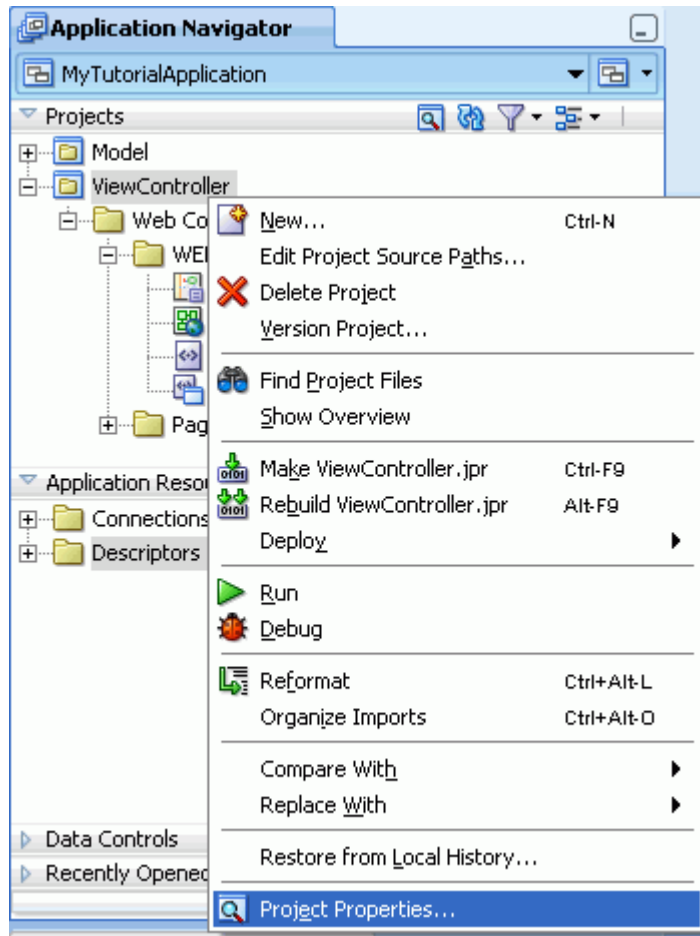


4. On the Application Name tab, in the Application Name field, enter `MyTutorialApplication`.
5. Click **Finish**. Oracle JDeveloper generates the base files, including two projects, for your application, which you can see in the Application Navigator (Figure 3–3):
 - **Model**, in which you define the JavaBeans and other data controls you need if the application is to perform any back-end logic.
 - **ViewController**, in which you'll create the JavaServer Faces (JSF) page that will consume WebCenter services and portlets.

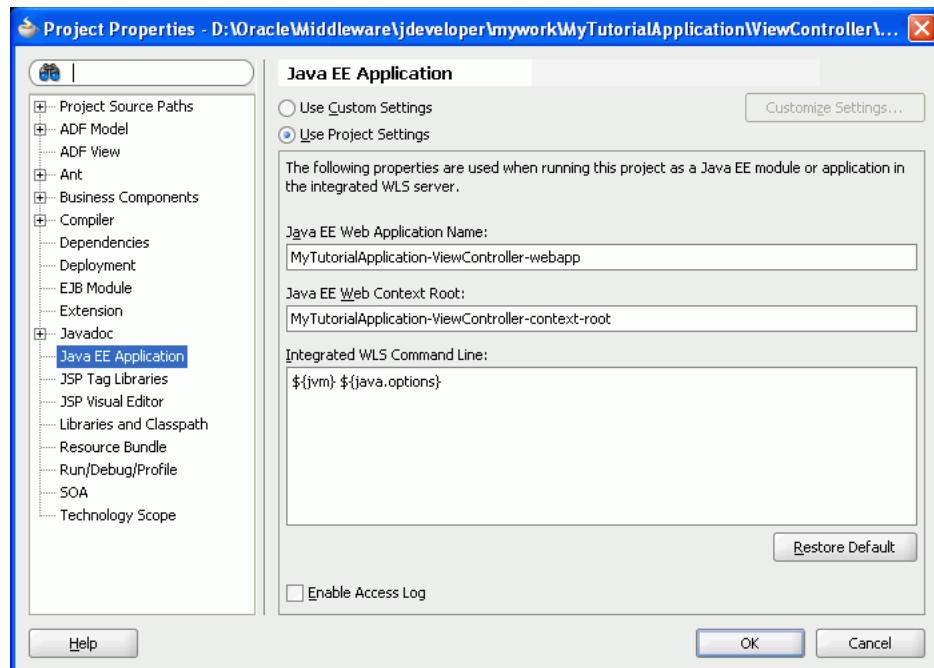
Figure 3–3 Generated Application Project Files in the Application Navigator

6. Let's make a few adjustments for the purposes of testing our application. Right-click the **ViewController** project, then choose **Project Properties** (Figure 3–4).

Figure 3-4 Editing the Project Properties



7. In the Project Properties dialog box, in the left column, choose **Java EE Application** (Figure 3-5).

Figure 3–5 Project Properties: Java EE Application

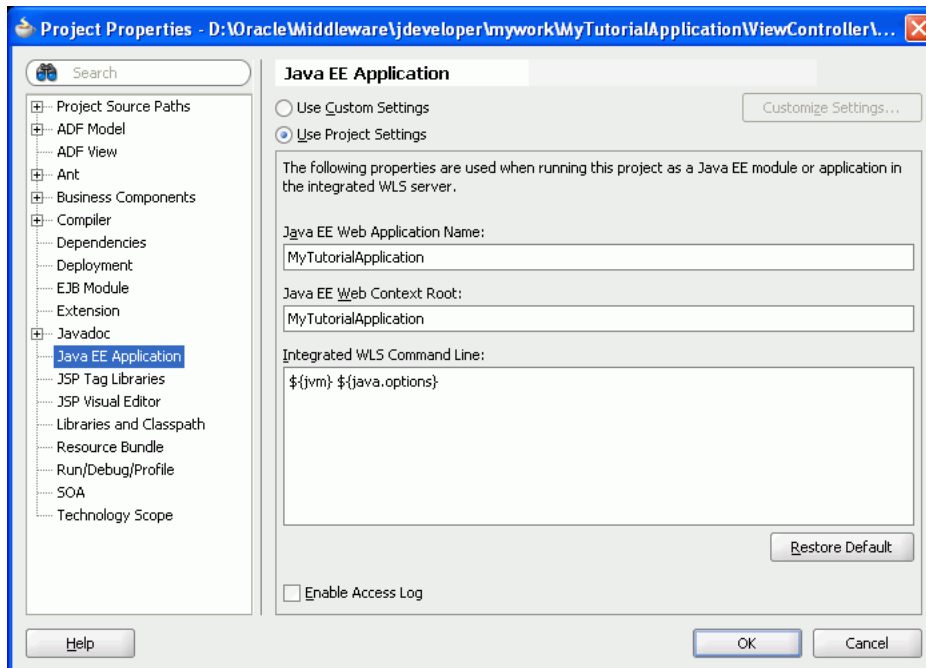
8. Here, we'll change the context root of the application to make it easier for us to reference the application resources that we'll use throughout this tutorial. In the Java EE Web Application Name field, enter:

MyTutorialApplication

9. In the Java EE Web Context Root field, enter the same value:

MyTutorialApplication

The Project Properties dialog box should now look like [Figure 3–6](#):

Figure 3–6 Project Properties with the Modified Context Root Values

10. Click **OK** to accept your changes.

11. Save your application by clicking the **Save All** icon in the toolbar.

Now that you have created a basic custom WebCenter application, you can now create a page. Before we create the page, though, let's add the image files to our application so that we can use them with our page.

Step 2: Add the Images Files to the Application

Now that we've created our application, let's quickly add the image files we want to use, including a logo for our page. Ensure that you've followed the steps in [Chapter 2, "Preparing for the Tutorial,"](#) which includes a step for obtaining the images files you will add.

To add the image files to our application:

1. In your file system directory (for example, Windows Explorer), navigate to the location where you installed Oracle JDeveloper (JDev_Home) and locate the following directory:

```
JDEV_USER_HOME\mywork\MyTutorialApplication\ViewController\public_html
```

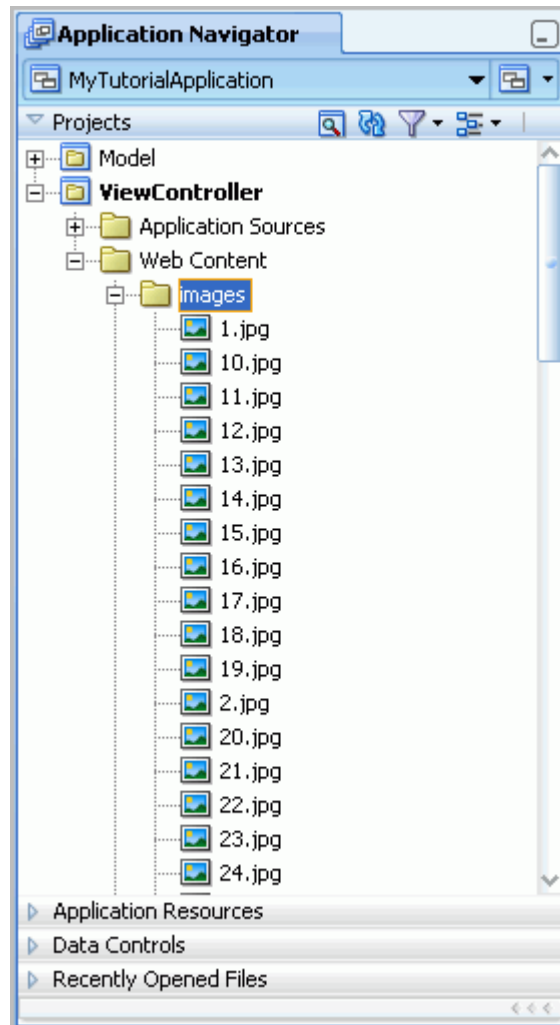
JDEV_USER_HOME refers to the default directory where JDeveloper stores your projects, and depends on how the JDEV_USER_HOME environment variable is set. This could be, for example, your C:\ drive, or it could be D:\Oracle\Middleware\, and so on. You should examine your file system to find out where this directory is set.

The MyTutorialApplication subdirectory was automatically generated when you created the application.

2. In the public_html directory, create a directory called images.

3. Locate the tutorial sample files you downloaded and extracted in [Chapter 2, "Preparing for the Tutorial,"](#) and copy the contents of the `C:\TutorialContent\images` folder into the new `images` directory.
4. Return to Oracle JDeveloper and click the **Refresh** icon next to the Projects list in the Application Navigator. You should now see the `images` folder in the Application Navigator ([Figure 3-7](#)).

Figure 3-7 Images in the Application Navigator



Note: You'll notice that the image names are enumerated: `1.jpg`, `2.jpg`, and so on. These image names match the product ID of the items we will retrieve when we create our portlets in [Chapter 5, "Building Portlets and Wiring Them in Your Application."](#) If you change the names of these image files, the statement may not return the correct results.

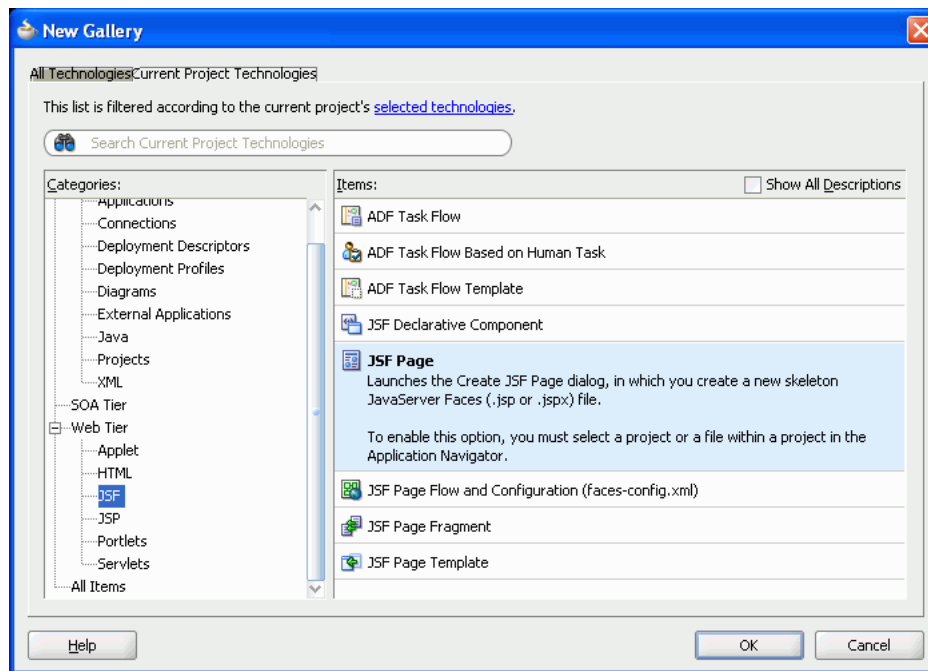
You can now use these images with your application.

Step 3: Create a Page

In this step, you will learn how to create a simple JSF page for your application, which will contain the services and portlets that you'll configure and add in the subsequent steps of this tutorial.

1. In the Application Navigator for `MyTutorialApplication`, right-click the **ViewController** project, then choose **New**.
2. In the New Gallery, under Web Tier, choose **JSF**.
3. Under Items, choose **JSF Page**, then click **OK** (Figure 3–8).

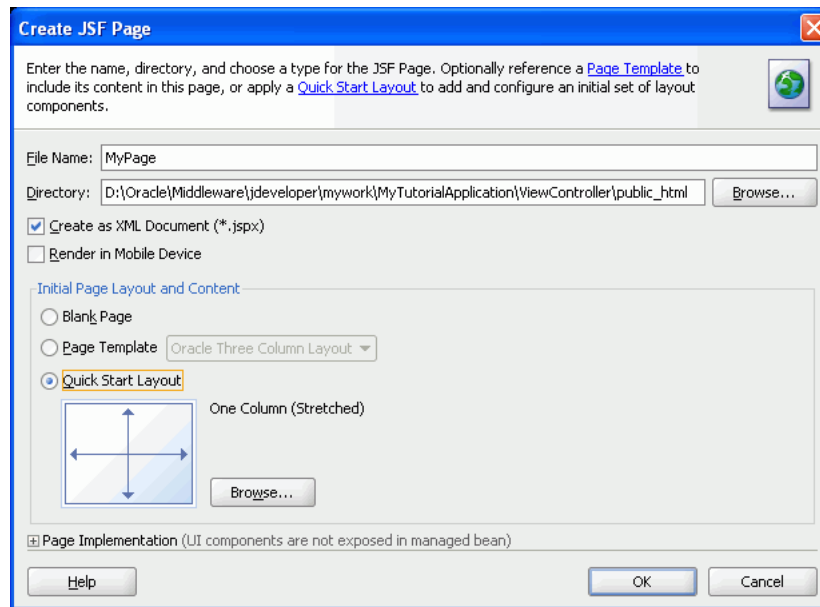
Figure 3–8 Choosing JSF Page from the New Gallery



4. In the Create JSF Page dialog box, in the Name field, enter `MyPage`.
5. Ensure the **Create as XML Document (*.jspx)** box is selected.
6. Let's set up an initial layout for our page. WebCenter Framework includes a few "quick start" layouts that help you get started with creating a page layout. You can use these layouts when you begin creating your own applications, or create your own layout from the beginning. In this tutorial, let's use a quick start layout.

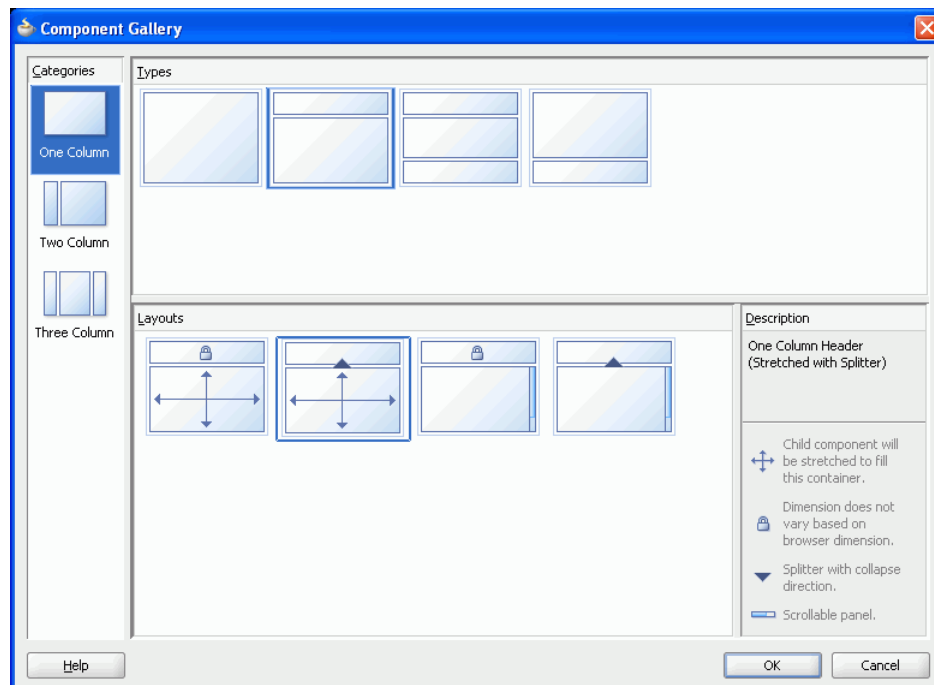
Under Initial Page Layout and Content, select **Quick Start Layout**, then click **Browse** (Figure 3–9).

Figure 3–9 Create JSF Page Dialog Box



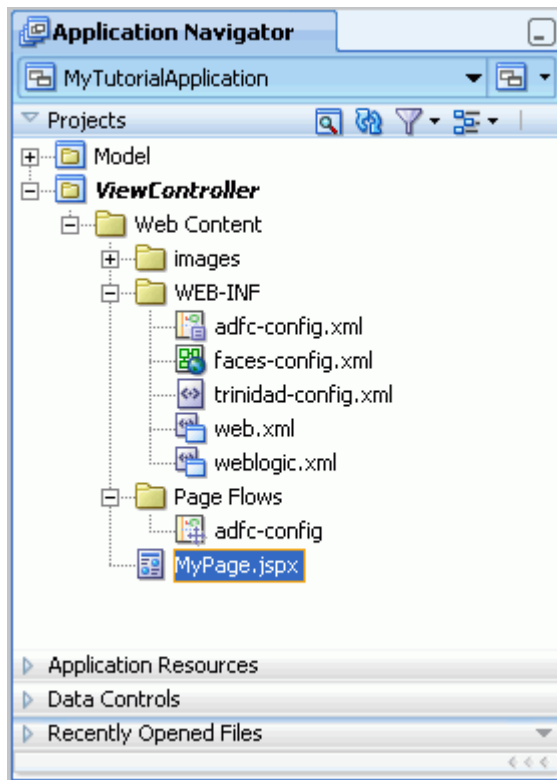
7. Under Categories, select **One Column**.
8. Under Types, select the second type.
9. Under Layouts, select the second layout. The Component Gallery should look like Figure 3–10.

Figure 3–10 Selecting an Initial Layout



10. Click **OK**, then click **OK** again.

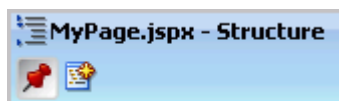
JDeveloper finishes the wizard and displays your page in the Application Navigator (Figure 3–11).

Figure 3–11 MyPage in the Application Navigator

11. To the right of the Application Navigator, you'll notice your page displays in the Design view (notice the Design tab is highlighted at the bottom of the view), as shown in [Figure 3–12](#).

Figure 3–12 Design tab

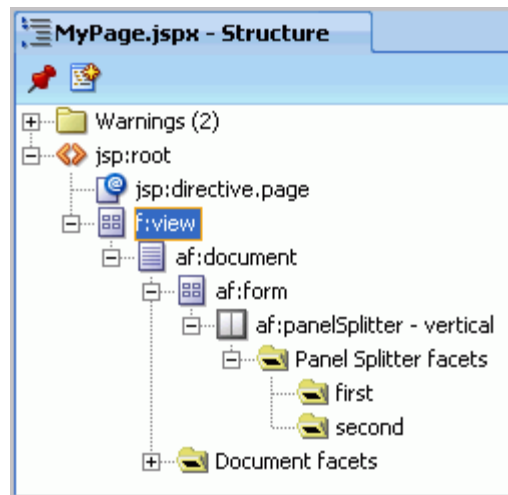
12. Below the Application Navigator, notice the Structure window for your page. This view shows all the elements of your page in a hierarchical view. Just below that Structure tab, you'll see a pushpin icon ([Figure 3–13](#)).

Figure 3–13 Pushpin in the Structure Window

Clicking this icon toggles the behavior of the Structure window -- if it is pressed, then the Structure window displays the current view no matter where you click in Oracle JDeveloper. If it is not pressed, the Structure window updates according to where you click in JDeveloper.

You can expand the nodes in this view to see the various components that were automatically added to your page, since you chose to use the Quick Start Layout. Notice the Panel Splitter and Panel Splitter facets under `f:view`, for example ([Figure 3–14](#)).

Figure 3–14 Structure Window for MyPage.jspx



Note: You can alternatively change to the Source view of the page. For the purposes of the tutorial, using the Structure window enables you to see clearly where you've added a new component.

Now that we've created our application and a JSF page, let's adjust the layout of our page and add Oracle Composer to our page to enable users to customize the page at runtime.

Step 4: Add Layout Components to the Page

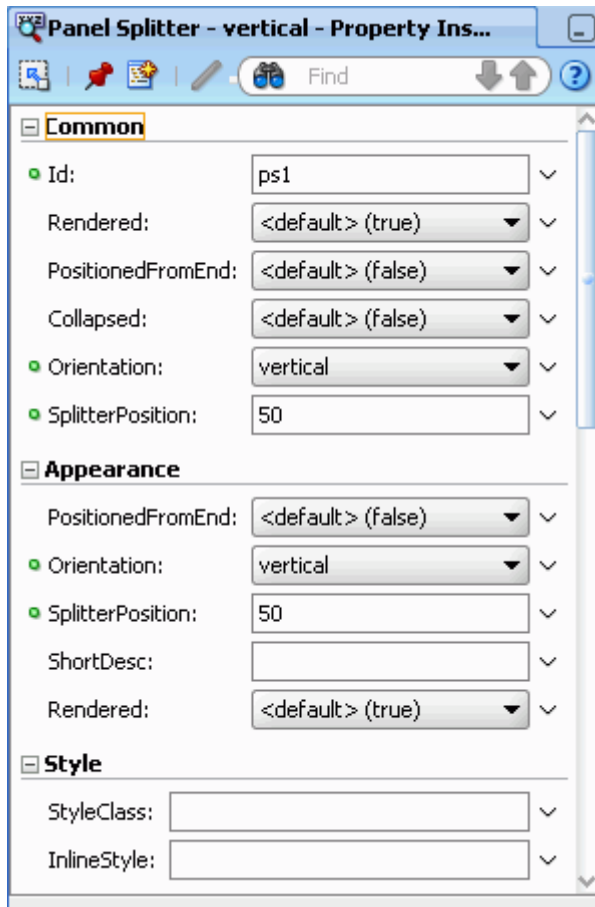
Selecting the Quick Start Layout when we created the page gave us a few "starter" layout components, specifically the Panel Splitter, which gives us the basic framework of our page. Now, let's add a few layout components to the page and structure the header section so that we can begin to add content, such as a logo image.

To add layout component to the page:

1. In the Structure window, select the **Panel Splitter**, which is listed as `af:panelSplitter - vertical`. You'll notice that the Property Inspector for this component displays to the right and below the Component Palette (Figure 3–15). The contents of the Property Inspector update depending on your focus in Oracle JDeveloper. Similar to the Structure window, you can use the pushpin icon in the Property Inspector to freeze the view according to the currently selected component.

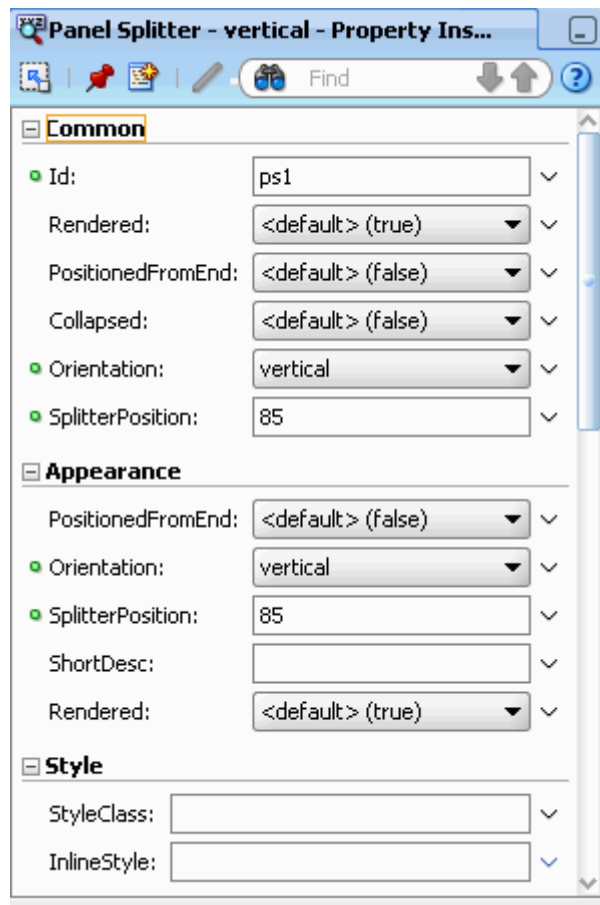
Note: You can always view the Property Inspector for a component by right-clicking the component, then choosing **Go To Properties** from the context menu.

Figure 3–15 Property Inspector for the Panel Splitter



2. In the Property Inspector, under **Common**, locate the **SplitterPosition** property, and change the value from 50 to 85 (Figure 3–16). Once you click elsewhere in JDeveloper, the property is saved.

Figure 3–16 Setting the SplitterPosition Property



3. In the Structure window, expand the **Panel Splitter**. You'll notice it contains two facets. In the next step, we'll drop an ADF Faces component onto the `first` facet.
4. Add an ADF Faces layout component to our page to control how the content will display.

In the Component Palette, choose **ADF Faces** from the list. If the Component Palette is not currently displaying, you can show it by choosing **View > Component Palette**.

Note: The Component Palette is context sensitive. That is, the contents of the Component Palette update according to the focus of your view in Oracle JDeveloper. As you're going through this tutorial, if you suddenly "lose" components in the Component Palette or do not see the components described, try ensuring that you have the correct page or panel selected.

5. Under Layout, scroll down to **Panel Stretch Layout**, select it (Figure 3–17), and drag and drop it onto the `first` facet in the Structure window.

Figure 3–17 Panel Stretch Layout in the Component Palette

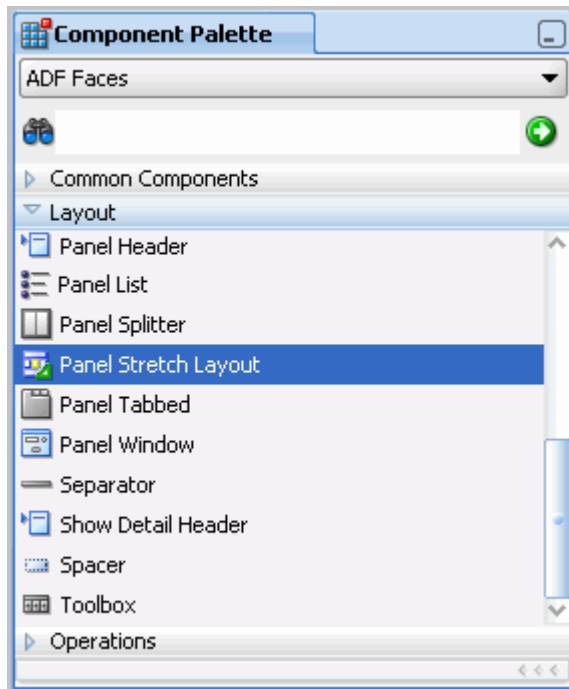
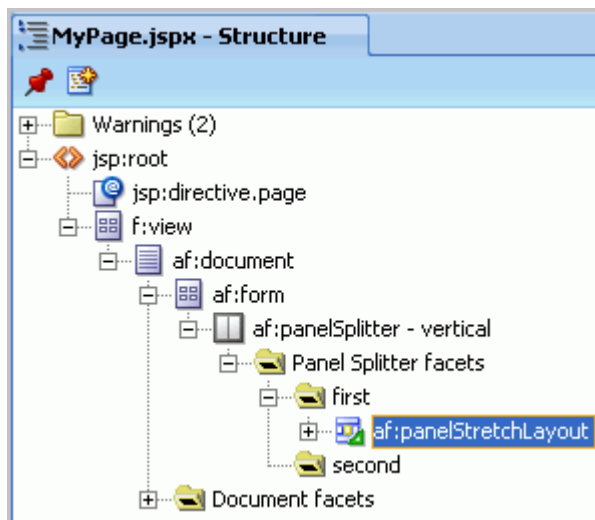
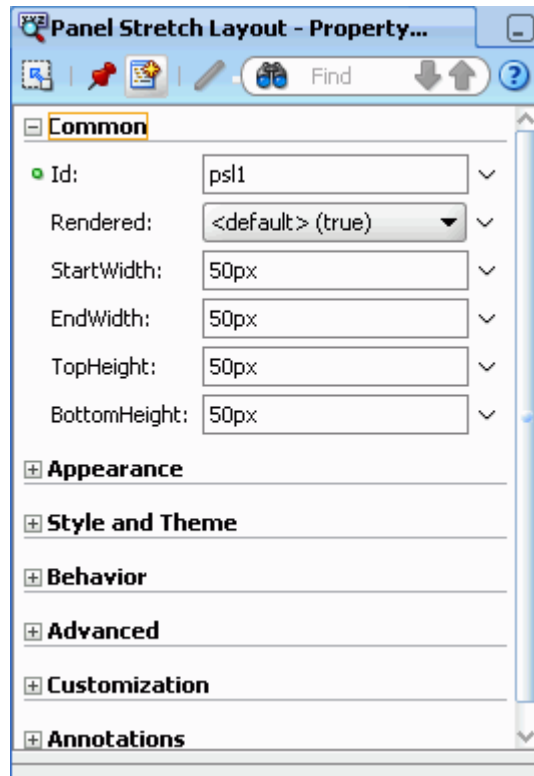


Figure 3–18 shows the Panel Stretch Layout in the Structure window.

Figure 3–18 Panel Stretch Layout in the Structure Window

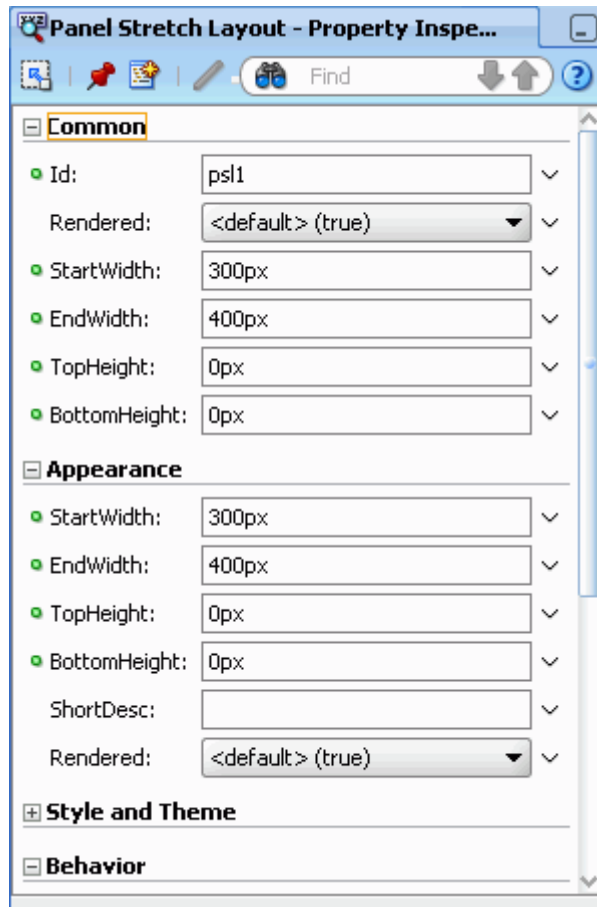


6. When you drop the Panel Stretch Layout component, you'll notice that the Property Inspector now displays the properties for this component (Figure 3–19).

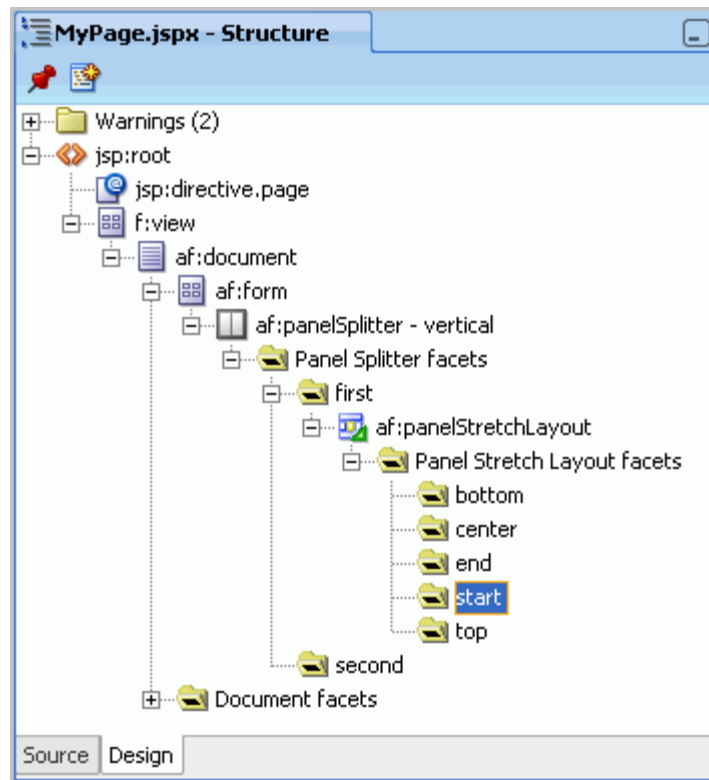
Figure 3–19 Properties of the Panel Stretch Layout

7. Set the following properties in the Property Inspector, as shown in [Figure 3–20](#):
 - **StartWidth:** 300px
 - **EndWidth:** 400px
 - **TopHeight:** 0px
 - **Bottomheight:** 0px

Figure 3–20 Property Inspector for the Panel Stretch Layout with Updated Values

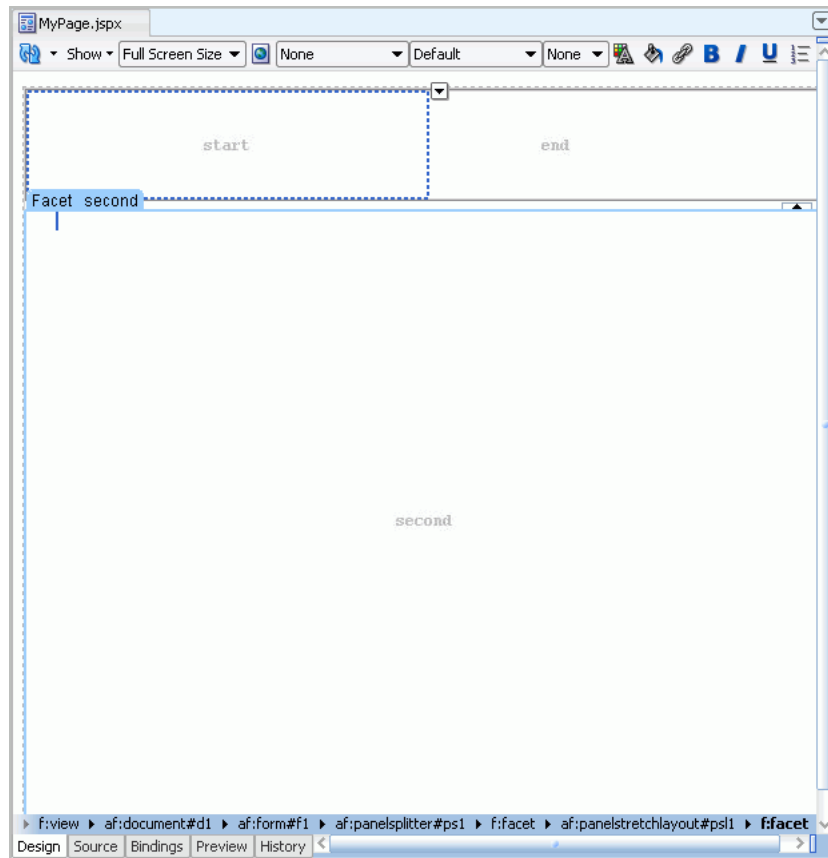


8. Next, let's add a logo to the header of the page, in the `start` facet of the Panel Stretch Layout. In the Structure window, expand the Panel Stretch Layout you just added so that you can see the different facets (Figure 3–21), then select the start facet.

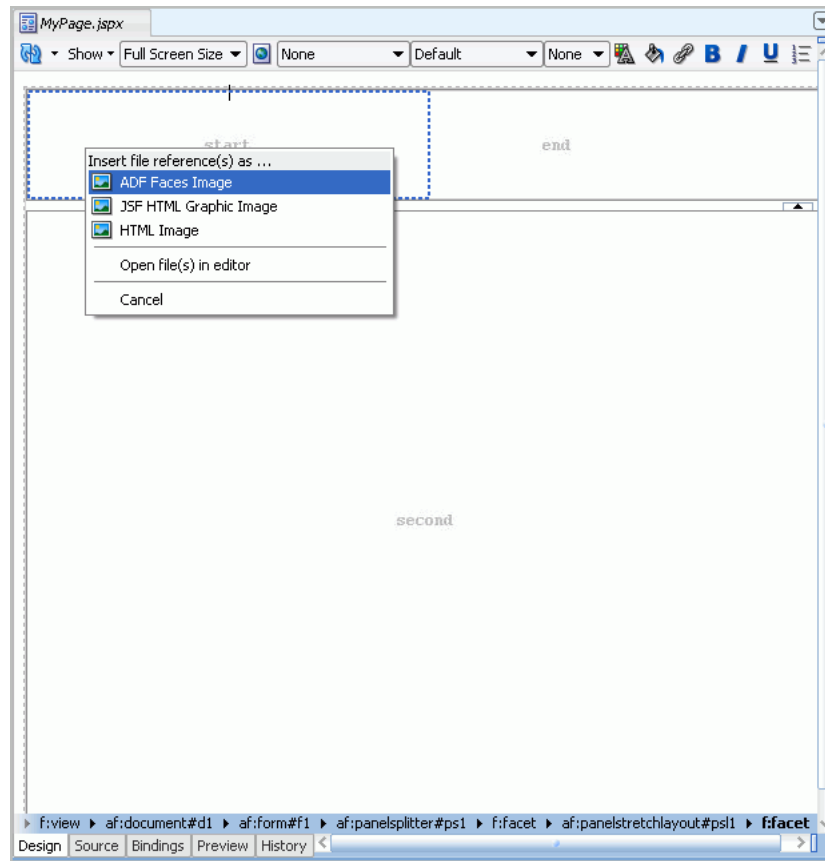
Figure 3–21 Panel Stretch Layout Facets

Notice how the corresponding facet is highlighted in the Design view (Figure 3–22). This is where we'll add the logo image.

Figure 3–22 Start Facet in the Design View

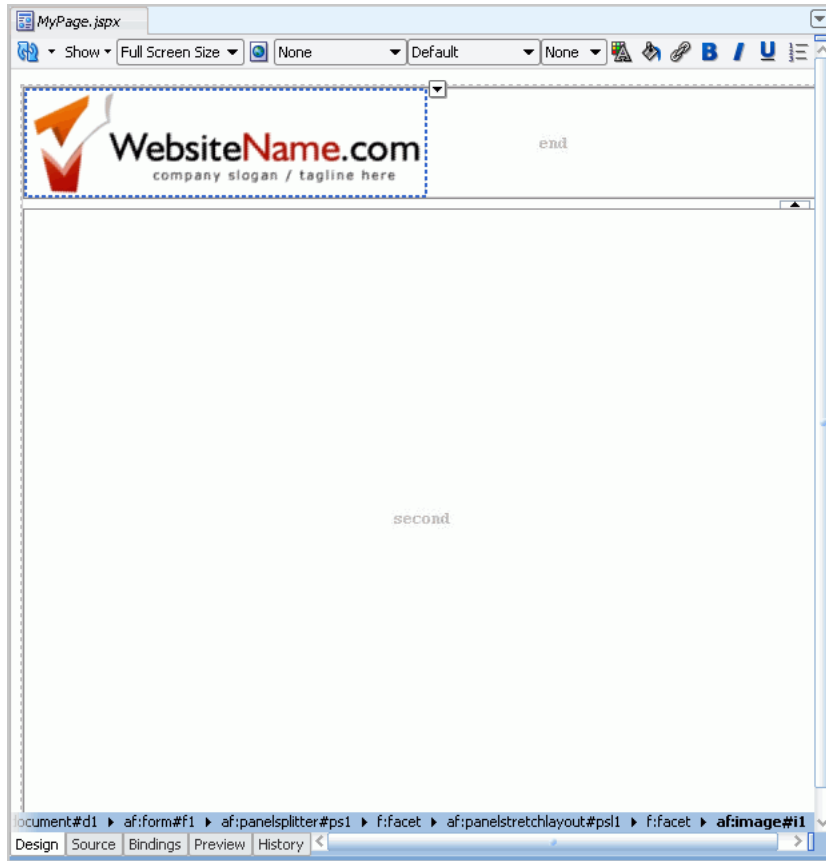


9. The logo we'll add is an image file we added to our application in [Step 2: Add the Images Files to the Application](#). In the Application Navigator, under **ViewController > Web Content**, open the **images** folder.
10. Drag and drop **logo.png** onto the start facet in the Design view of your page. When you drop the image, choose **ADF Faces Image** from the context menu ([Figure 3–23](#)).

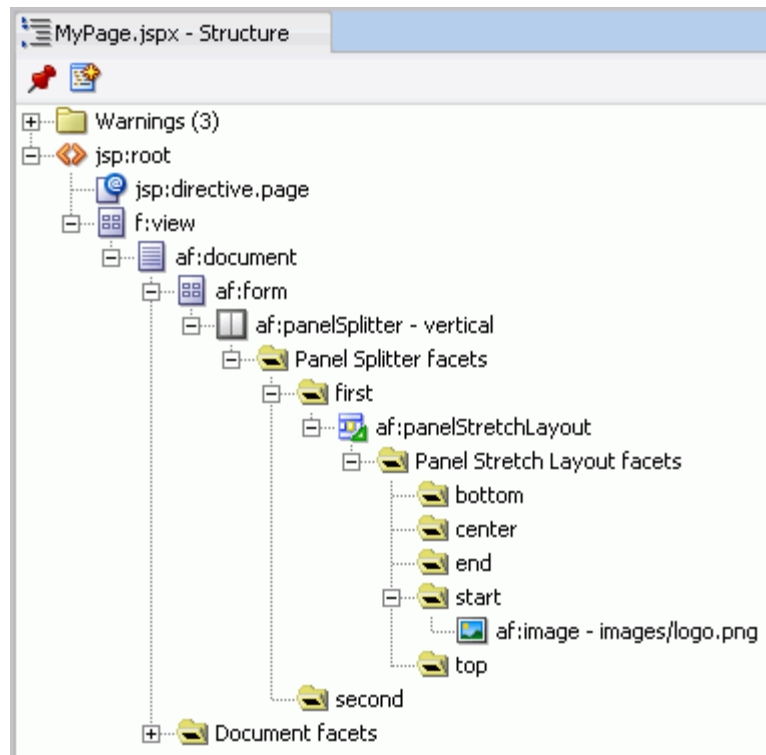
Figure 3–23 Choosing ADF Faces Image from the Context Menu

The logo displays on your page, as shown in [Figure 3–24](#).

Figure 3–24 Logo Image on Page in the Start Facet



Also notice that the image now displays in the Structure window, as shown in [Figure 3–25](#).

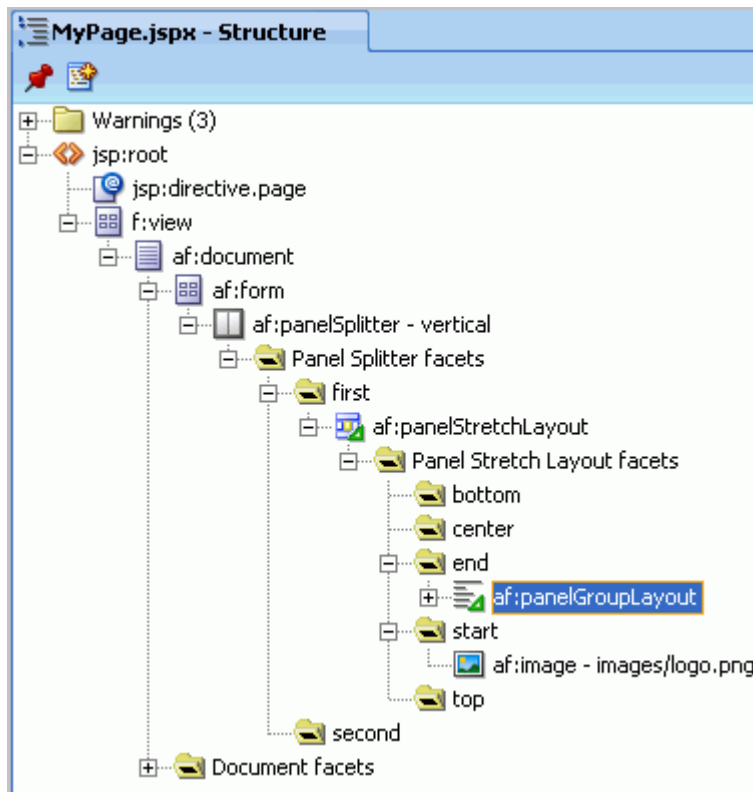
Figure 3–25 Logo Image in the Structure Window

- Let's finish the layout of the header. In the Design view, notice the end facet (Figure 3–26).

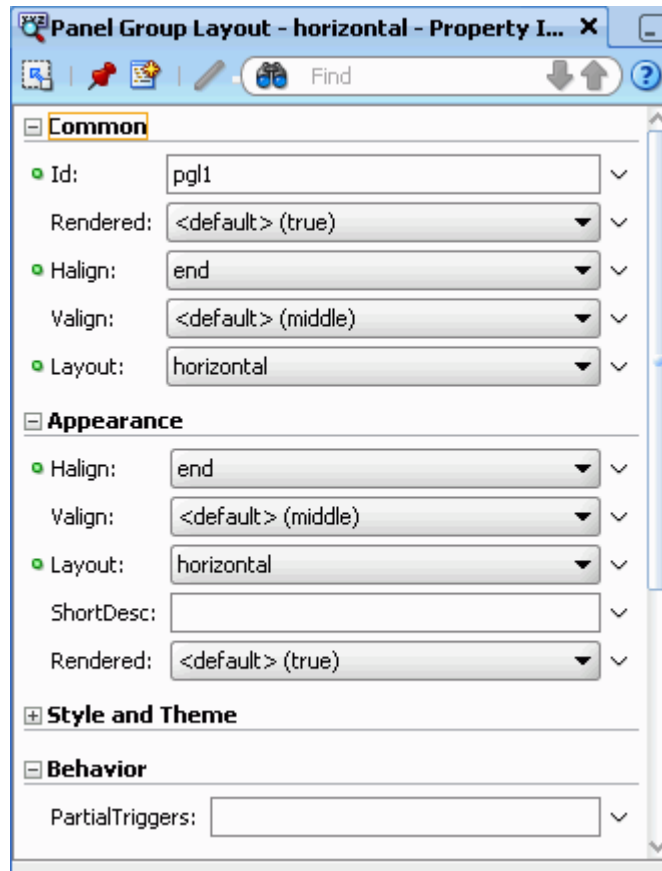
Figure 3–26 End Facet in the Design View

From the Component Palette, drag and drop the **Panel Group Layout** ADF Faces component onto this facet. You can see this now either in the Design view or, more easily in the Structure window (Figure 3–27).

Figure 3–27 Panel Group Layout in the End Facet



12. While the Panel Group Layout is selected, in the Property Inspector, under Appearance, change the **Halign** property to **end**. This changes the alignment of the components you will add to this layout component.
13. Change the **Layout** property to **horizontal** (Figure 3–28).

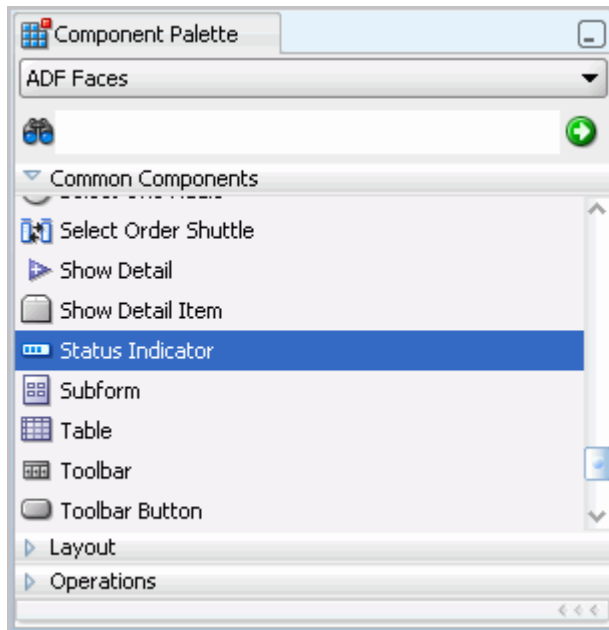
Figure 3–28 Changing the Properties of the Panel Group Layout

14. For the purposes of this tutorial, let's add a Status Indicator. The Status Indicator keeps us informed of the application's activity during runtime. For example, if you click a link, the Status Indicator will let you know the application is accessing the target of that link.

Ensure that the **MyPage.jspx** tab is displaying in the Design view.

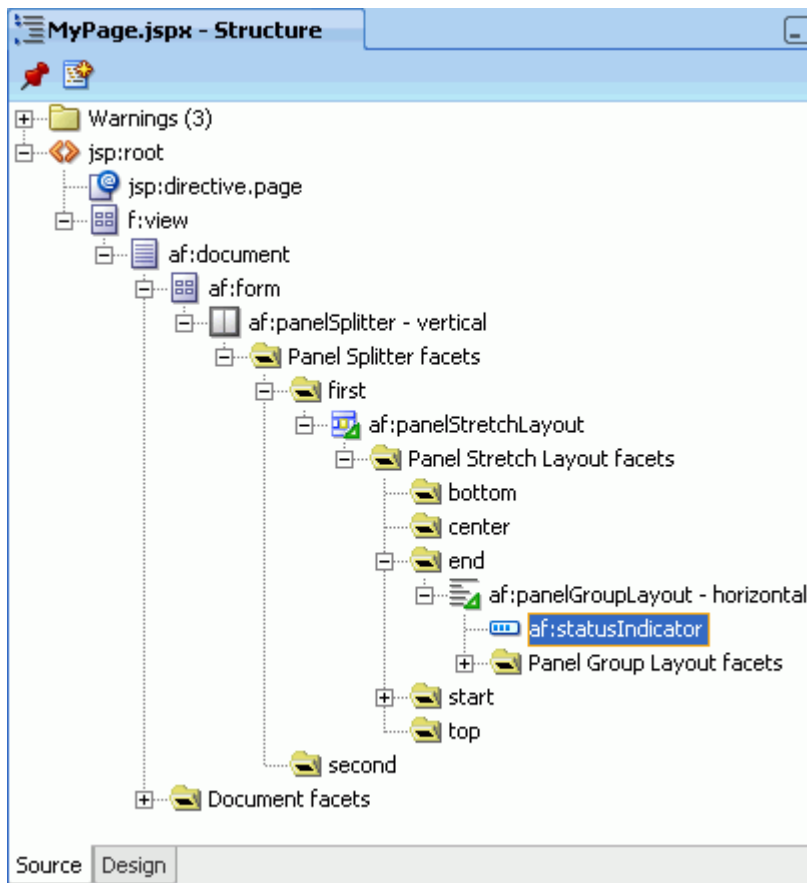
15. In the Component Palette, choose **ADF Faces** from the list to display the ADF Faces components.
16. Under Common Components, scroll down the list and locate **Status Indicator**, then select it (Figure 3–29).

Figure 3–29 Status Indicator in the Component Palette



17. Drag and drop the **Status Indicator** onto the Panel Group Layout in the end facet, as shown in [Figure 3–30](#).

Figure 3–30 Status Indicator on the Panel Group Layout



18. Let's examine how the page looks at runtime. Right-click the page in the Design view, then choose **Run**. The page containing the logo and status indicator displays in your browser, as you can see in [Figure 3–31](#).

Figure 3–31 *MyPage with Logo and Status Indicator at Runtime*



19. Return to JDeveloper.

Now that we've set up the initial header for our page, let's add Oracle Composer so that we can customize our page at runtime.

Step 5: Add Oracle Composer to the Page to Enable Customization

In traditional Java EE applications, if you wanted to edit pages (for example, add content, edit security definitions, and so on), you had to make these changes in Oracle JDeveloper, which is the application design time, and then redeploy the updated application to the production environment. With Oracle Composer, you and your application users can now edit your pages at runtime and see the results of your modifications immediately.

Using Oracle Composer, you can give users the ability to move objects around on their page, hide or show content, as well add new content to the page. Let's add Oracle Composer to our page.

To add Oracle Composer to our page:

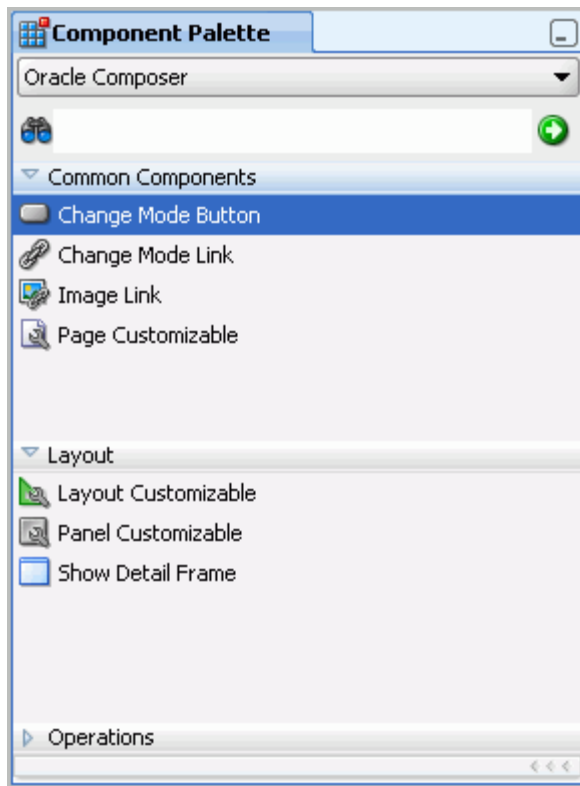
1. First, let's add the Change Mode Button to the page. This button will enable runtime users of your application to switch between viewing the page and editing it using Oracle Composer.

In the Component Palette, choose **Oracle Composer** from the list at the top.

2. Under Common Components, drag and drop the **Change Mode Button** component onto the Panel Group Layout in the Structure window. Adding this component will let you switch back and forth between the Edit mode and the View mode of your page at runtime. [Figure 3–32](#) shows the Change Mode Button in the Component Palette.

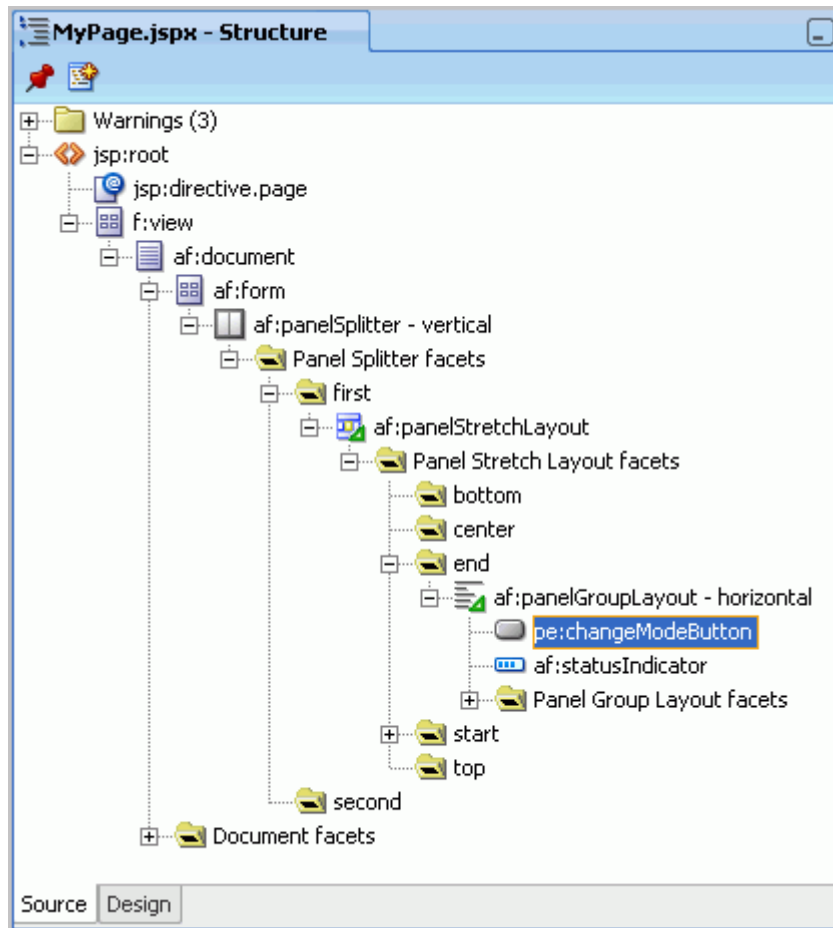
Note: You can adjust the display of the Component Palette to see all of the options. You can do so by clicking and dragging the bars (such as the Layout bar) up and down to view the component names.

Figure 3–32 Change Mode Button in the Component Palette



3. If necessary, you can move the button above the Status Indicator by dragging it in the Structure window ([Figure 3–33](#)).

Figure 3–33 Change Mode Button in the Structure Window



4. Add a Page Customizable component to the layout. The Page Customizable, which is an Oracle Composer component, adds the runtime customization capabilities to the page.

In the Component Palette, ensure that **Oracle Composer** is still selected.

5. Under Common Components, drag and drop **Page Customizable** (Figure 3–34) onto the `second` facet in the Structure window (Figure 3–35).

Figure 3-34 Page Customizable in the Component Palette

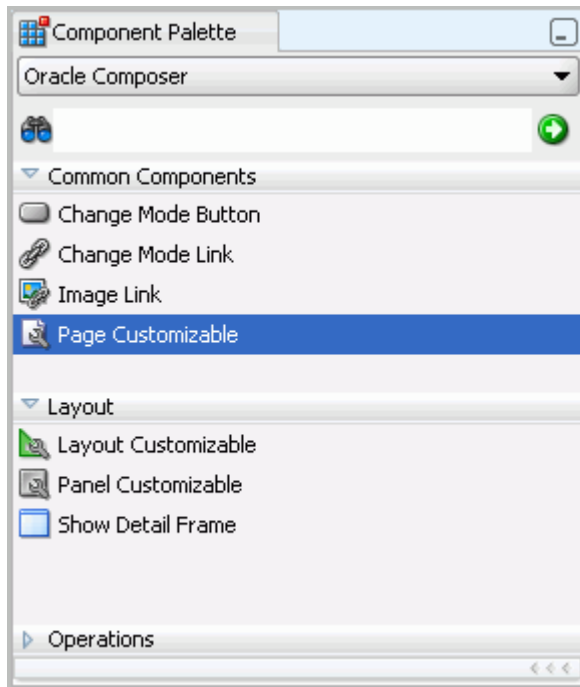
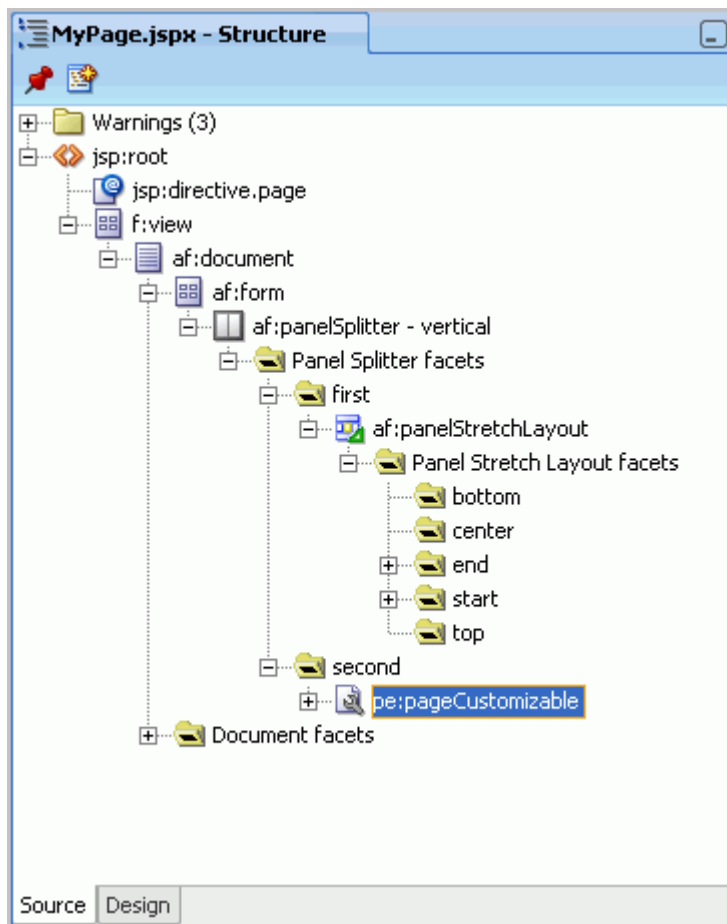
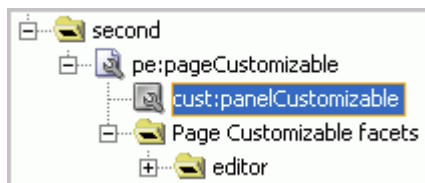


Figure 3-35 Page Customizable in the Second Facet



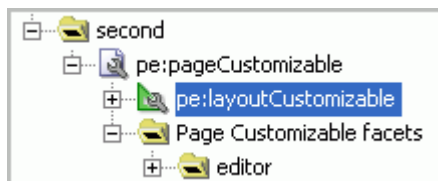
6. In the Structure window, expand the Page Customizable, then delete the **Panel Customizable** since we do not need it (Figure 3-36). You can delete it by selecting it, then pressing your Delete key.

Figure 3-36 *Deleting the Panel Customizable*



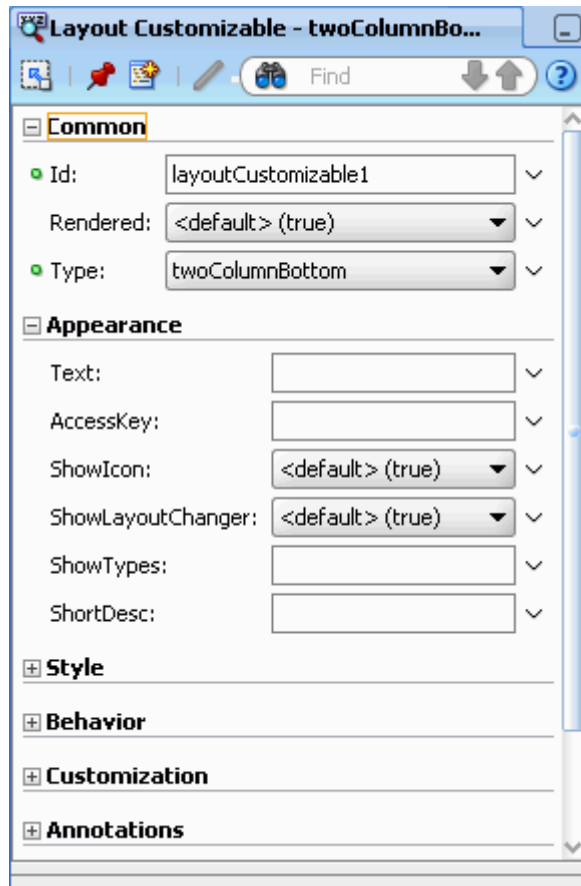
7. Now, we'll add a Layout Customizable to our page to create a layout for this region. In the subsequent chapters, we will add services and portlets to this area. From the Component Palette, drag and drop **Layout Customizable** onto the **Page Customizable** in the Structure window (Figure 3-37).

Figure 3-37 *Layout Customizable in the Page Customizable*



8. While the Layout Customizable is selected, in the Property Inspector, under Common, set the **Type** property to **twoColumnBottom** (Figure 3-38).

Figure 3–38 Setting the Layout Customizable Properties



9. Now that we've added a few Oracle Composer components to our page, let's run the page and check out some of its features at runtime.

Right-click **MyPage** in the Design view, then choose **Run** from the context menu to view your page at runtime (Figure 3–39).

Figure 3–39 MyPage at Runtime with the Change Mode (Edit) Button



We'll explore Oracle Composer at runtime in the next step.

Step 6: Customize the Page at Runtime Using Oracle Composer

After you add Oracle Composer to a page at design time (in JDeveloper), you can run your page and customize the page at runtime (in a web browser). Your application users can also customize their pages at runtime -- this way, you do not have to go back to JDeveloper every time you want to modify the appearance of your application.

In this step, we'll test a few features that Oracle Composer offers. But, for a more in-depth description of using Oracle Composer, refer to the Oracle Fusion Middleware User's Guide for Oracle WebCenter.

1. While MyPage displays in your browser, you'll notice an Edit button next to the Status Indicator. Click the **Edit** button to see how the page looks in Edit mode (Figure 3-40).

Figure 3-40 Edit Mode of MyPage



2. Let's take a quick tour of what you can do in Edit mode.

In the top horizontal region, click **Add Content** to view the runtime catalog. Here, you can click **Open** to see the different ADF Faces Components you can add at runtime (Figure 3-41).

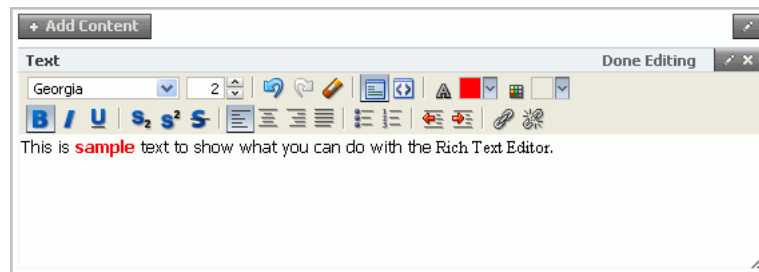
Figure 3–41 Adding Content at Runtime



3. Let's add a Text object to see how it looks. In the Catalog, next to Text, click **Add**, then click **Close**. A text box now displays in the region, as shown in [Figure 3–42](#).

Figure 3–42 Text Box in the Edit Mode at Runtime

4. In the upper right corner of the region, click **Edit Text** to switch to the Rich Text Editor.
5. Place your cursor in the text box and enter some sample text. You can play around with the different functions in the toolbar, as well, such as changing the text to bold or switching the font, as shown in [Figure 3–43](#).

Figure 3–43 Entering and Modifying Text in the Rich Text Editor at Runtime

6. When you're finished, click **Done Editing**.
7. You can change the overall layout of your page by clicking **Change Layout**, then choosing a layout option ([Figure 3–44](#)).

Figure 3–44 Changing the Layout at Runtime



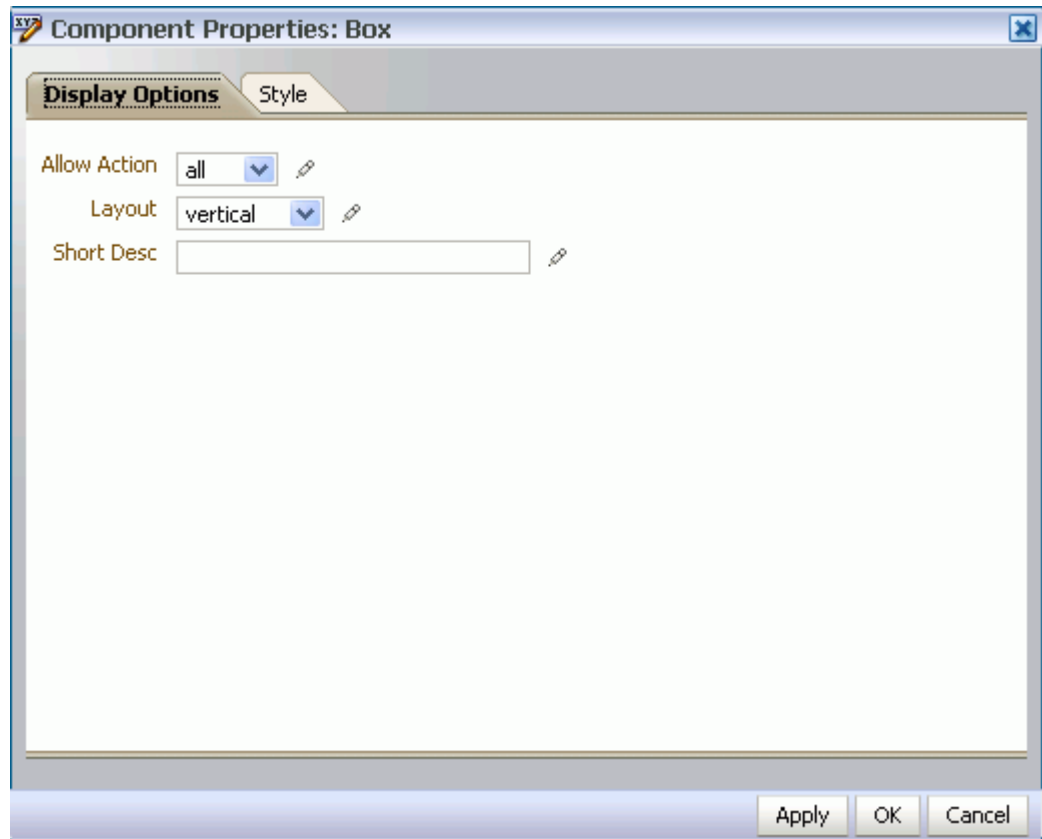
8. Try selecting a layout option, such as the three column layout. The page automatically refreshes with your changes, as you can see in [Figure 3–45](#).

Figure 3–45 MyPage in a Three Column Layout at Runtime



9. You can also change the layout of each of the component boxes by clicking the pencil icon in the upper right corner of the box ([Figure 3–46](#)).

Figure 3–46 Component Properties: Box



In the Component Properties for the layout box, you can change the layout of the box, the background color, and so on. Click **Cancel** to exit the Properties pane.

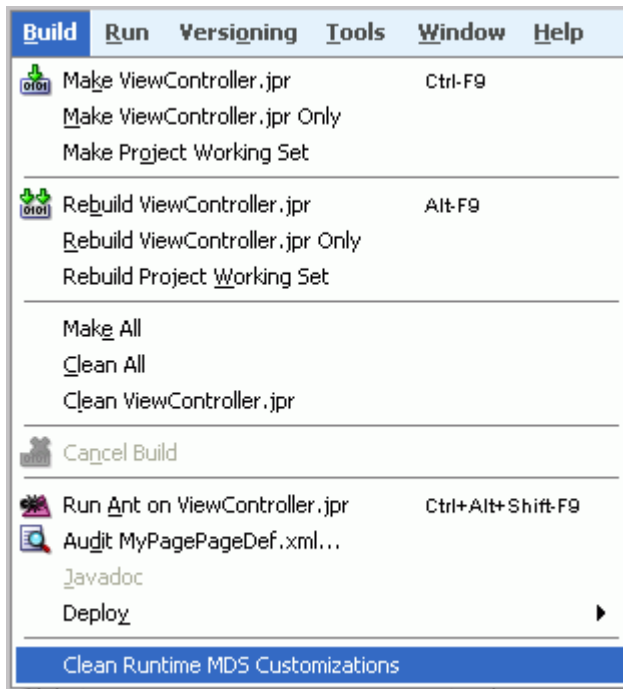
10. You can perform several other tasks at runtime, as well. For example:
 - Drag and drop components from one region to another. Hover your mouse over the text box, and while you see the crosshairs, click and drag the text box from one region to another on the page.
 - Delete components. In the upper right corner of the box, click the **X** to delete the component.
11. Return to Oracle JDeveloper.

Notice that the changes you made, for example the sample text, does not display in your Design view. Modifications that you or your application users make at runtime in Oracle Composer are stored behind the scenes, which means that the changes you make do not affect the source of your application.

12. Although your application view is not affected by runtime modifications, the customizations made at runtime are stored in the MDS (Metadata Services). So, while you are developing an application, it is good practice to clear out these changes.

From the Build menu, choose **Clean Runtime MDS Customizations** (Figure 3–47). If a confirmation dialog box displays, click **Yes**.

Figure 3–47 Clean Runtime MDS Customizations Menu Option



You can learn more about Oracle Composer and Oracle Metadata Services in the Oracle Fusion Middleware Developer's Guide for Oracle WebCenter.

Now that we've created a custom WebCenter application and customizable JSF page, and taken a quick tour of Oracle Composer at runtime, let's start adding services and components to our page in the next chapter, [Chapter 4, "Adding WebCenter Web 2.0 Services to Your Application."](#)

Adding WebCenter Web 2.0 Services to Your Application

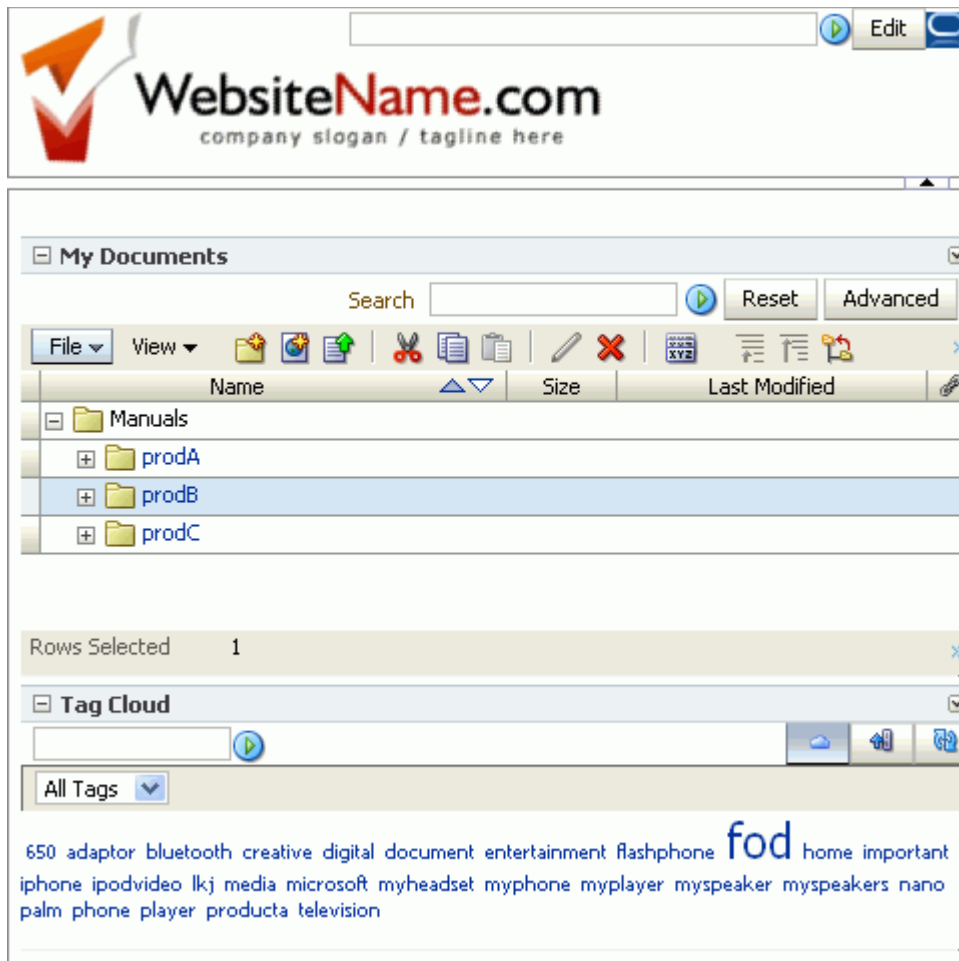
Oracle WebCenter Framework enables you to enhance your application with WebCenter Web 2.0 features, such as secure information sharing and online collaboration. In this chapter, we will explore a few of the features you can use to add these capabilities to your application.

In this lesson, you will add a search toolbar, document library, and tagging capabilities to your application. With the document library, or the Documents service, you can enable your users to share content like Word documents or PDFs in an easy-to-navigate environment, right in the application. The Tags service enables you and your users to add keywords to documents, files, and other items in your application to make it easier to search and organize content. This service also includes a “tag cloud” task flow where you can visualize the mapping of the keywords to your application contents right on your page.

These services are just a few examples of the myriad services Oracle WebCenter Suite offers. You will see how easy it is to use services to make your application dynamic and provide frequently-used communication options to your users. To learn more about WebCenter Web 2.0 Services, refer to the Oracle Fusion Middleware Developer's Guide for Oracle WebCenter.

[Figure 4-1](#) shows how your page will look at the end of this lesson.

Figure 4–1 MyPage.jspx at the End of this Lesson



Introduction

This lesson contains the following steps:

- [Step 1: Add the Search Toolbar Task Flow to the Application](#)
- [Step 2: Create a Connection for the Documents Service](#)
- [Step 3: Add the Document Library Task Flow to Your Application](#)
- [Step 4: Browse Documents at Runtime](#)
- [Step 5: Create a Database Connection to the WebCenter Schema for the Tags Service](#)
- [Step 6: Add the Tags Service to Your Application](#)
- [Step 7: Use, Add, and Search Tags in Your Application at Runtime](#)

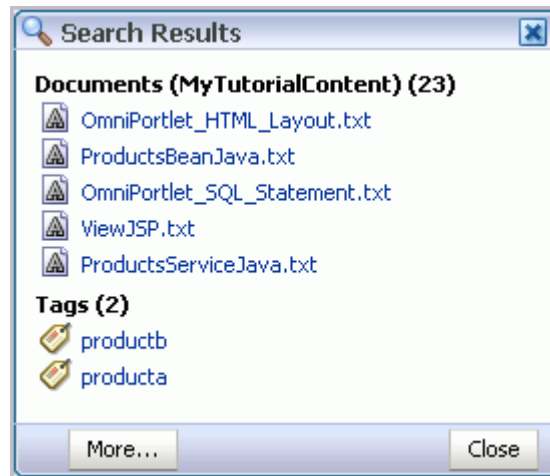
Before you begin the steps in this lesson, ensure you have followed the steps up to this point in the tutorial.

Step 1: Add the Search Toolbar Task Flow to the Application

With Oracle WebCenter Framework, you can also allow your users to search through information within the services. For example, you may want to add a Search field so

that users can search for a particular file in your document library. [Figure 4–2](#) shows a sample of the search results you can retrieve in your application.

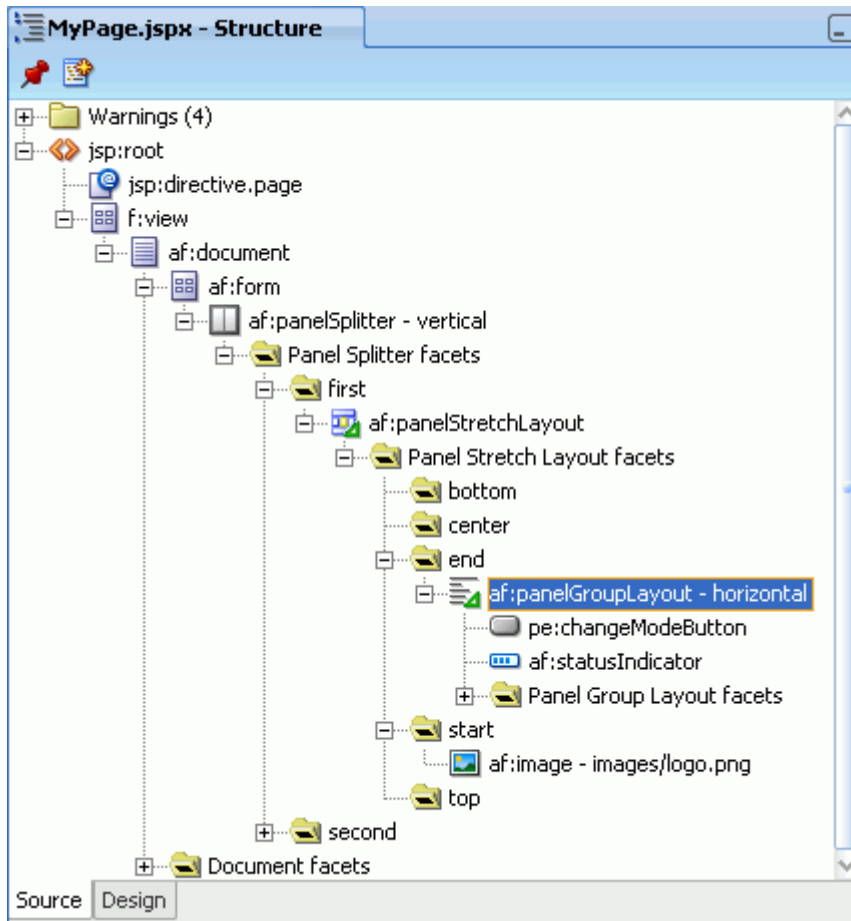
Figure 4–2 Sample Search Results



To add the Search service to your application:

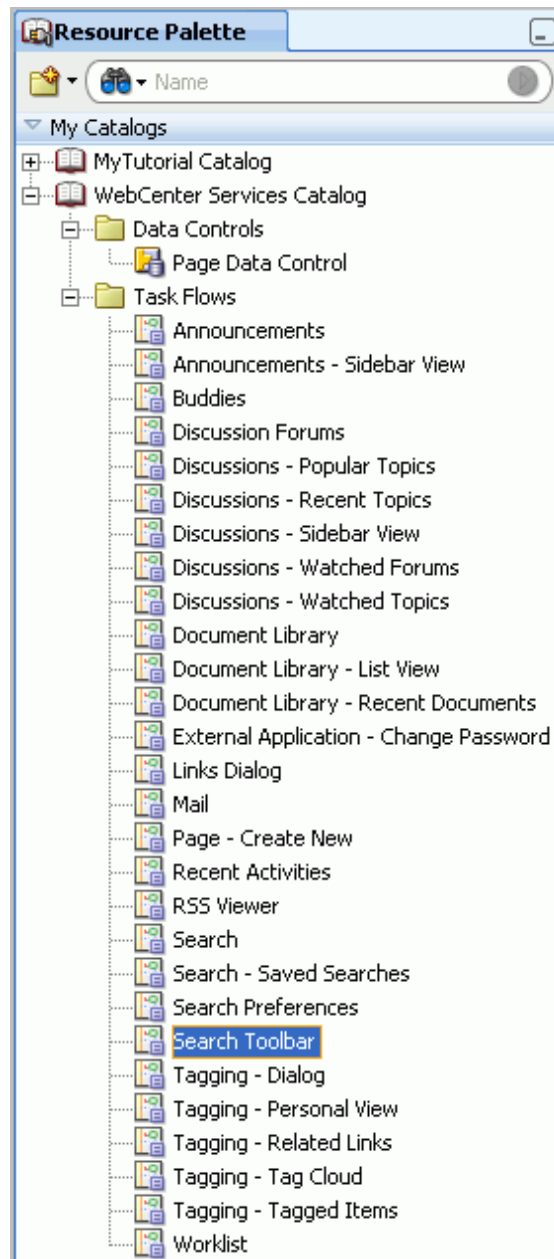
1. Make sure your page, `MyPage.jspx`, is open.
2. In the Structure window, locate the Panel Group Layout (in the end facet) that you added in [Chapter 3, "Creating a WebCenter Application with a Customizable Page."](#) [Figure 4–3](#) shows the Panel Group Layout selected. You will add the Search task flow here.

Figure 4-3 Panel Group Layout



3. As we mentioned in [Chapter 3, "Creating a WebCenter Application with a Customizable Page,"](#) you can use the pushpin in the Structure window to freeze the current view. For this step, you click **MyPage** in the Design view, then ensure the pushpin is in the "freeze" position (pressed). This way, when you click the Search toolbar task flow in the ensuing steps, the Structure window does not contextually refresh.
4. In the Resource Palette, in the WebCenter Services Catalog, open the **Task Flows** node. If you do not see the Resource Palette in Oracle JDeveloper (it usually displays as a tab next to the Component Palette), choose **View > Resource Palette**.
5. Under My Catalogs, open the **WebCenter Services Catalog**, then expand **Task Flows**.
6. Locate the **Search Toolbar** task flow ([Figure 4-4](#)).

Figure 4-4 Search Toolbar Task Flow in the WebCenter Services Catalog

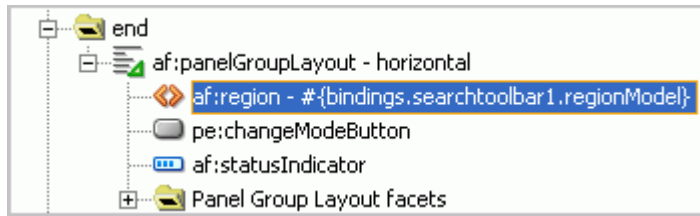


Drag and drop this task flow onto the Panel Group Layout then choose **Region** from the context menu. If you are prompted to add the appropriate libraries, choose **Add Libraries**.

7. In the Structure window, move the **Search Toolbar** task flow (`af:region - #{bindings.sarchtoolbar1.regionModel}`) above the Change Mode Button.

Note: As previously mentioned, you can use the pushpin in the Structure window to freeze the current view. For this step, you should click **MyPage** in the Design view, then ensure the pushpin is in the "freeze" position (pressed).

Figure 4-5 Search Toolbar in the Structure Window



8. Run the page to your browser to see how it looks at runtime. [Figure 4-6](#) shows the Search toolbar next to the Edit (Change Mode) button.

Figure 4-6 Search Toolbar on MyPage at Runtime



9. Now that we've added a Search toolbar to our application, let's add the Documents service so that we have content to search.
Return to JDeveloper.

Step 2: Create a Connection for the Documents Service

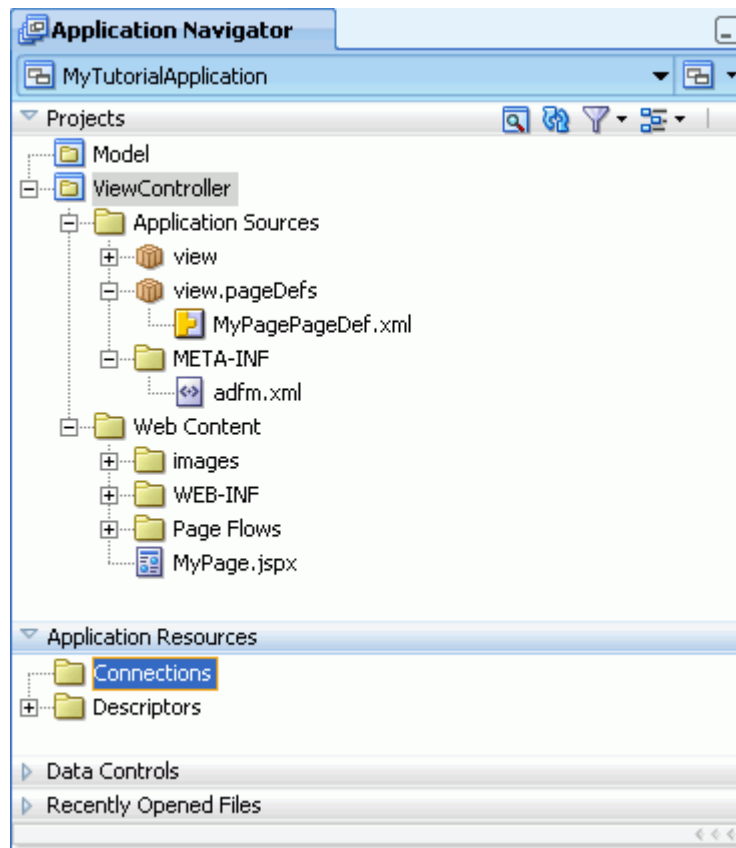
With every service in the WebCenter Framework, we perform a few basic steps: configure the service with our application, add the service task flow to our page, then run the page and customize or use the service at runtime.

Before we can take advantage of the Documents service, which enables us to browse, manage, and create documents at runtime, we must first create a connection to our content repository. In this tutorial, our content repository will be the directory we created in [Chapter 2, "Preparing for the Tutorial."](#)

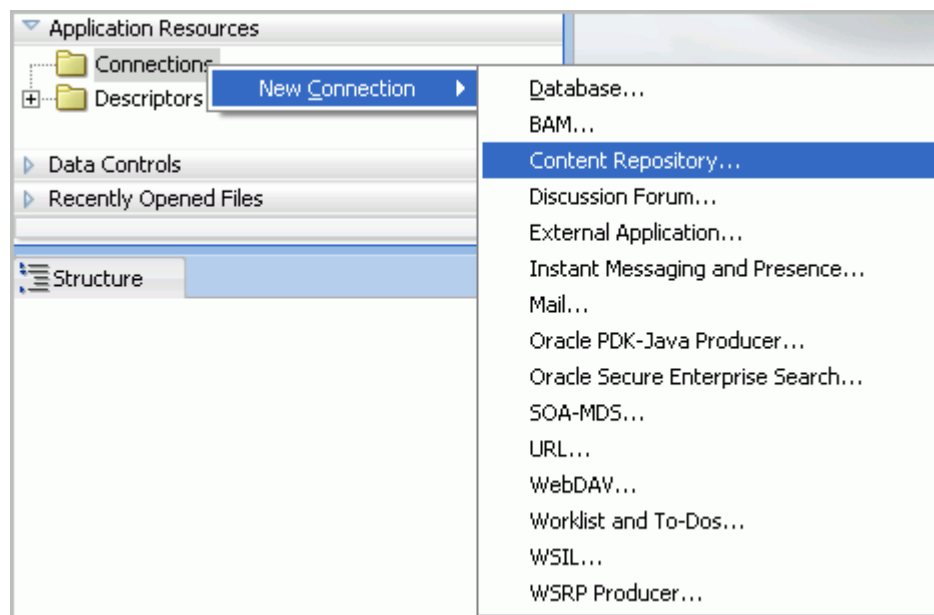
You can learn more about creating connections for the Documents service in the Oracle Fusion Middleware Developer's Guide for Oracle WebCenter.

To create a connection to our content repository:

1. In Oracle JDeveloper, in the Application Navigator, locate the Application Resources node and expand it. Notice the **Connections** folder here ([Figure 4-7](#)).

Figure 4-7 Connections Folder in the Application Resources Panel

2. Ensure the **ViewController** project is highlighted.
3. Under Application Resources, right-click the **Connections** folder, then choose **New Connection > Content Repository...** (Figure 4-8).

Figure 4-8 Choosing the New Content Repository Connection Menu Option

4. In the Create Content Repository Connection dialog box that displays, notice the first option: you can choose to either limit the usage of this connection to the current application by choosing **Application Resources**, or enable all applications created using your instance of Oracle JDeveloper to use this connection by choosing **IDE Connections**.

Select **Application Resources**.

Note: Creating a connection here enables you to reuse the same connection throughout your application. If you chose **IDE Connections**, you could reuse the connection in any application you create using your instance of Oracle WebCenter Framework. The connection name would display in the Resource Palette under **IDE Connections**.

5. In the Connection Name field, enter `MyTutorialContent`.
6. From the Repository Type list, choose **File System**.
7. Select **Set as primary connection for Documents service** to set this connection as the active connection any time you add the Documents service to your application.

If you did not choose this option and created a connection to another content repository, Oracle WebCenter Framework would automatically set the first connection you created as the active connection.
8. Under Configuration Parameters, let's set the Base Path to the location where you downloaded the tutorial sample files on your C drive (in [Chapter 2, "Preparing for the Tutorial"](#)), by entering `c:\TutorialContent`.
9. Click **Test Connection**. You should see a Success! message in the Status field, as shown in [Figure 4-9](#).

Figure 4-9 Create Content Repository Connection

Choose Application Resources to create a content repository connection owned by and deployed with the current application (MyTutorialApplication.jws). Choose IDE Connections to create a connection that can be added to any application.

Create Connection In: Application Resources IDE Connections

Connection Name:

Repository Type:

Set as primary connection for Documents service

Configuration Parameters (* = required):

Parameter	Value
* Base Path	c:\TutorialContent

Login Timeout (ms):

Authentication: Identity Propagation

External Application: + ✎

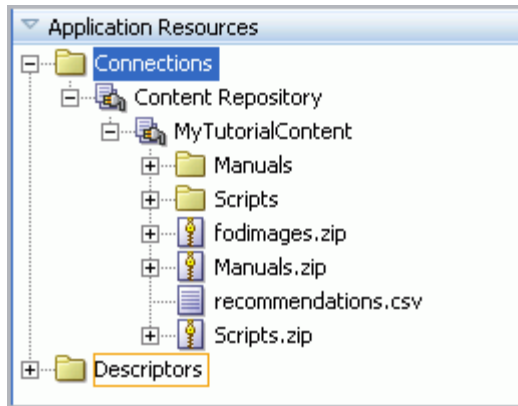
Specify login credentials for the current JDeveloper session:

User Name:

Password:

10. Click **OK**.

11. In the Application Navigator, in the Application Resources panel, you will see your new connection, as shown in [Figure 4-10](#).

Figure 4–10 New Connection in the Application Resources Panel

Note: To learn more about configuring and using WebCenter Services, refer to the Oracle Fusion Middleware Developer's Guide for Oracle WebCenter.

Step 3: Add the Document Library Task Flow to Your Application

Oracle WebCenter Framework enables content integration by:

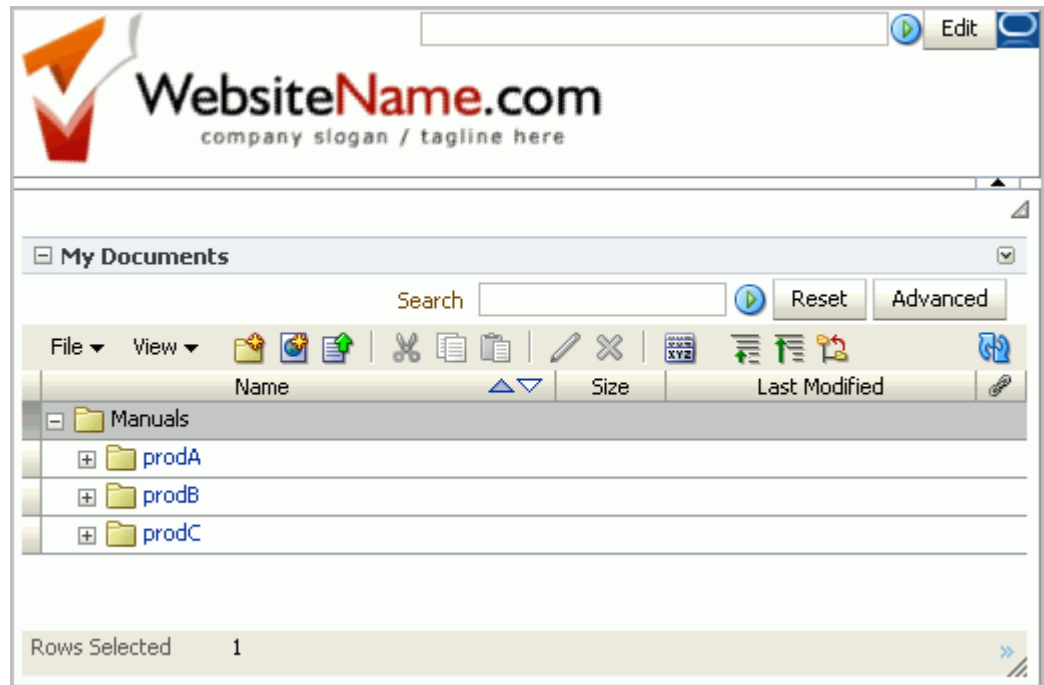
- Content Repository data controls: enable read-only access to a content repository, and maintain tight control over the way the content displays in the application.
- Documents service: enable users to view and manage documents in your organization's content repositories.

Both methods of integration use content repository connections, which you created in the previous step. In this tutorial, we'll use the Documents service to integrate content (the sample content on our file system that we downloaded in [Chapter 2, "Preparing for the Tutorial"](#)) into our application.

Using the Documents service, users can view, upload, and collaborate around documents. In this tutorial, you will use a file system connection that will let you view documents and enable your application users to share them, but not upload them. Using a content repository such as Oracle Content Server or Oracle Content Database, you can take advantage of all the features of this service. To learn more about using different content repositories and more details about the various Documents service task flows, refer to Chapter 15, "Integrating the Documents Service" in the Oracle Fusion Middleware Developer's Guide for Oracle WebCenter.

In this step, you will add the task flow provided by the service to your application. The Documents service provides several task flows, one of which is the Document Library. The Document Library task flow, when used with a file system content repository, enables your users to view folders and files within the application much as they would on their own file system in a read- and write-only format ([Figure 4–11](#)).

Figure 4–11 Document Library View of Sample Folders at Runtime

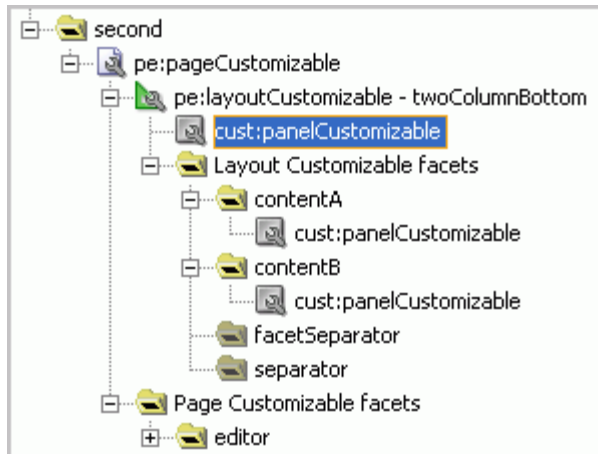


To add the Document Library task flow to your application at design time in Oracle JDeveloper:

1. In Oracle JDeveloper, ensure **MyTutorialApplication** is open and that you have configured your application for services and set up a connection to the content repository, as described in [Step 2: Create a Connection for the Documents Service](#).
2. Let's add the Document Library task flow to the page. Ensure that `MyPage.jspx` is open.
3. In the Structure window, locate where we are going to add the task flow ([Figure 4–12](#)). Under the second facet navigate to **pe:pageCustomizable > pe:layoutCustomizable - twoColumnBottom**. Here, you will see the Panel Customizable where we will add the task flow.

Note: As previously mentioned, you can use the pushpin in the Structure window to freeze the current view. For this step, you should click **MyPage** in the Design view, then ensure the pushpin is in the “freeze” position (pressed).

Figure 4–12 Location Where We Will Add the Document Library View

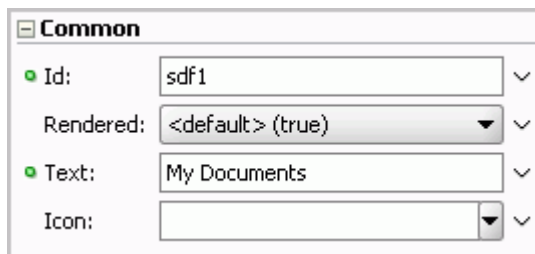


4. Add a chrome around the Document Library, so that at runtime, the layout shows a frame around the list of documents.

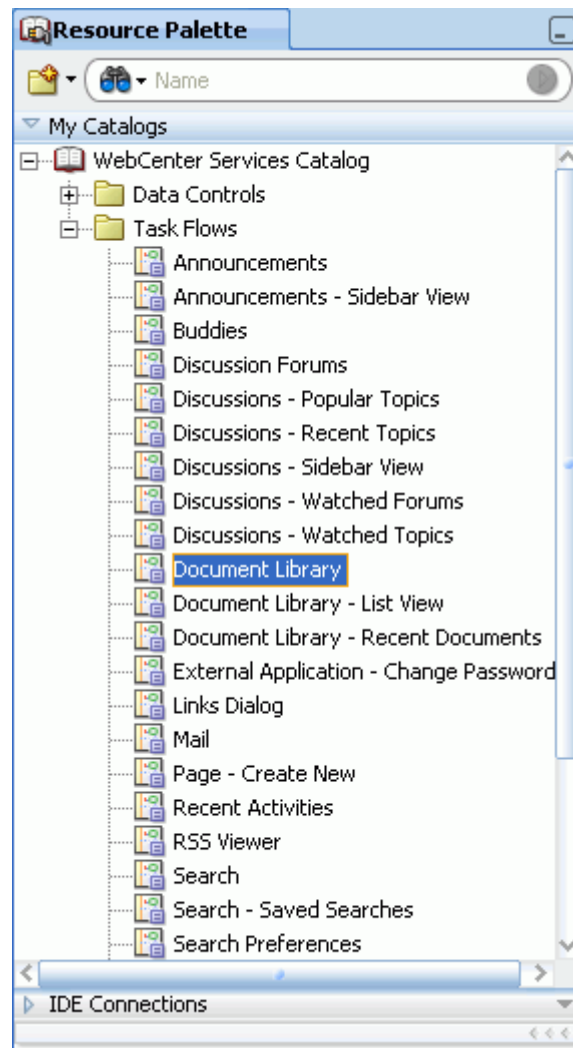
In the Component Palette, choose **Oracle Composer** from the list.

5. Under Layout, drag and drop **Show Detail Frame** onto the **cust:panelCustomizable** that you selected (as shown in [Figure 4–12](#)).
6. While the Show Detail Frame is selected, in the Property Inspector, under Common, change the **Text** property to `My Documents`, as shown in [Figure 4–13](#). Notice also that the frame is also selected in the Design view of your page.

Figure 4–13 Changing the Title of the Show Detail Frame

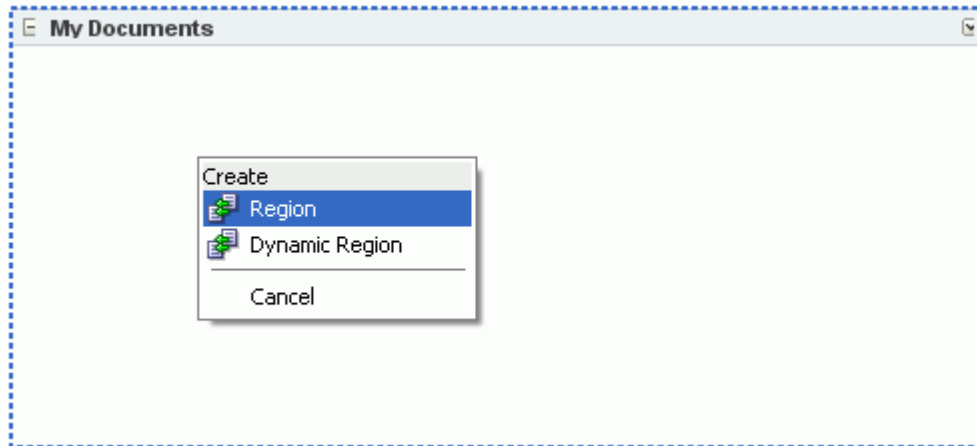


7. In the Resource Palette, under WebCenter Services Catalog, open the **Task Flows** folder and locate **Document Library**. [Figure 4–14](#) shows the task flow in the WebCenter Services Catalog.

Figure 4–14 Document Library Task Flow in the Resource Palette

8. Drag and drop the **Document Library** task flow onto the Show Detail Frame. You can do this either by dragging it onto the Show Detail Frame in the Structure window or in the Design view of the page.
9. In the context menu that displays, click **Create Region**, as shown in [Figure 4–15](#).

Figure 4–15 Dropping the Document Library Task Flow onto the Page in the Design View



10. If the Confirm Add ADF Library dialog box displays, click **Add Library**.
11. The Edit Task Flow Binding dialog box displays. In this dialog box, you can set up the connection parameter for the task flow.

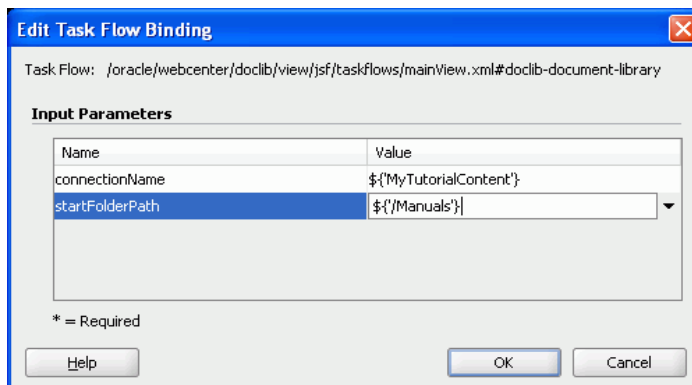
Click the **connectionName** parameter and enter `${'MyTutorialContent'}` to tell the Document Library task flow to use this connection.

Note: If you do not enter a connection name here, the content repository connection where you selected **Set as primary connection for Documents service** is the active connection.

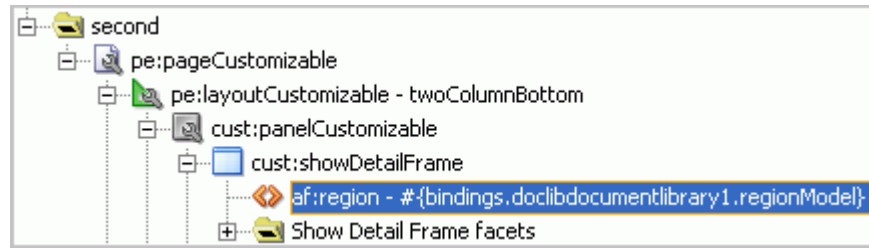
12. Since the connection points to the root folder of our tutorial sample files, let's update the startFolderPath parameter to the folder containing the product manuals.

Click the **startFolderPath** parameter and enter `${' /Manuals'}` next to it, then click **OK**. [Figure 4–16](#) displays the Edit Task Flow Binding dialog box with the appropriate values.

Figure 4–16 Edit Task Flow Binding Dialog Box



The Structure window shows the new task flow with the `af:region` tag, as shown in [Figure 4–17](#).

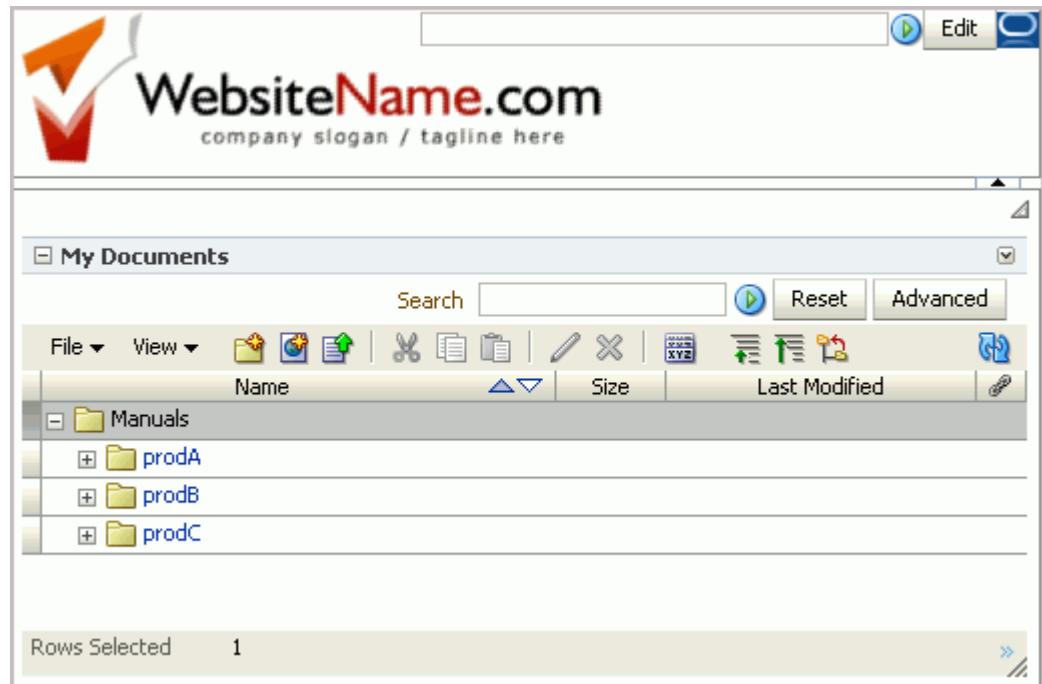
Figure 4–17 Document Library View Task Flow in the Structure Window

13. In the Application Navigator, right-click the **MyPage.jspx** page and select **Run**. In the next step, we'll examine the Document Library at runtime.

Step 4: Browse Documents at Runtime

Now that you've added the task flow to your application, you can see the documents in your content repository.

1. After you run your application to your browser, you'll see a list of your files display (Figure 4–18).

Figure 4–18 Documents in the Document Library View

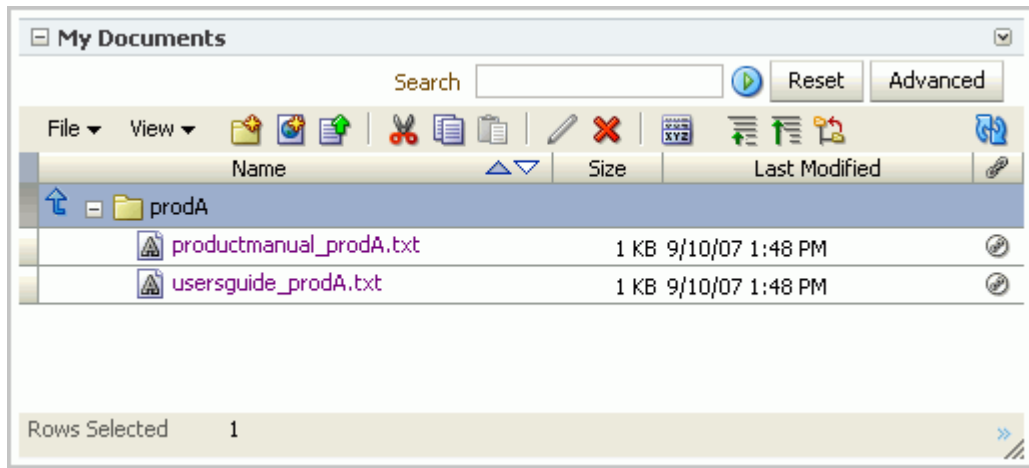
Let's examine this view. As you can see, the service shows by default the names of the folders: `prodA`, `prodB`, `prodC`. This information all comes from your file system and let's you quickly show your content without much coding.

Also notice the Search field within the My Documents frame. This search feature is limited to just the files in the document library. The search toolbar you added in [Step 1: Add the Search Toolbar Task Flow to the Application](#) searches across any page and service you add to your application.

2. Click the folder **prodA**. Notice how the folder opens within the context of the view. You can use the breadcrumbs to navigate throughout your document

library. You can also view the various attributes about the documents (for example, Size, Last Modified).

Figure 4–19 Document Library View Showing the Contents of a Folder



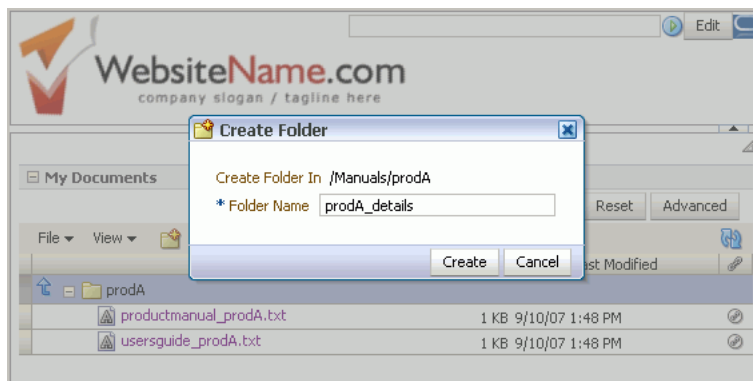
3. Click the **Create New Folder** icon in the toolbar (Figure 4–20).

Figure 4–20 Create New Folder Icon in the Toolbar

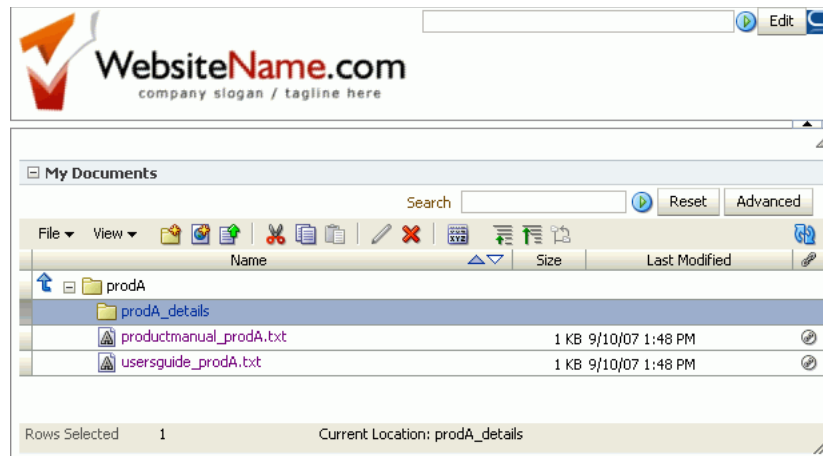


4. In the Create Folder dialog box, enter a folder name, such as `prodA_details` (Figure 4–21).

Figure 4–21 Document Library View with the Create Folder Dialog Box



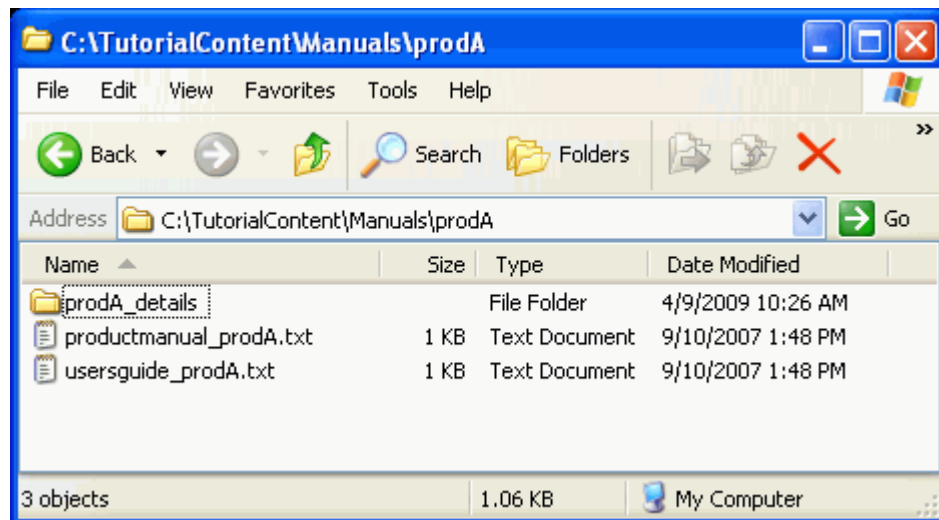
5. Click **Create**. Figure 4–22 shows the new folder in your browser.

Figure 4–22 *prodA_details* Folder in the Document Library at Runtime

- Since you are using your own file system as your content repository, any actions you perform at runtime also affect the content repository itself. Let's examine the content repository, in this case on your file system.

Notice the `prodA_details` folder also exists in your

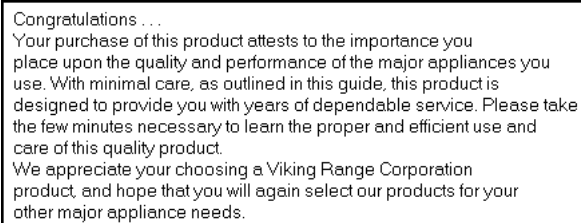
`C:\TutorialContent\Manuals\prodA` directory (Figure 4–23).

Figure 4–23 *New Folder in the File System*

While we're able to do this on our file system, if you use a secure repository like Oracle Content Server, you'll need appropriate privileges to perform these actions.

- Return to your browser to view the Document Library at runtime.
- Click the file `productmanual_prodA.txt`. Notice that a new browser tab or window displays with the contents of the text file (Figure 4–24).

Figure 4–24 Text File in a Browser Window



Congratulations . . .
Your purchase of this product attests to the importance you place upon the quality and performance of the major appliances you use. With minimal care, as outlined in this guide, this product is designed to provide you with years of dependable service. Please take the few minutes necessary to learn the proper and efficient use and care of this quality product.
We appreciate your choosing a Viking Range Corporation product, and hope that you will again select our products for your other major appliance needs.

You can easily navigate from the parent folder to the text file by switching your browser window. Document Library displays the content of text and image files in the browser. For more information on using the Document Library at runtime, refer to the Oracle Fusion Middleware User's Guide for Oracle WebCenter.

9. Return to Oracle JDeveloper.

As you can see, the Document Library view provides a quick and easy way for you to display content in a meaningful way and with functionality. You can learn more about this simple view and the more complex Document Library views in the Oracle Fusion Middleware Developer's Guide for Oracle WebCenter.

To exercise more direct control over the behavior, look, and feel of your content, you can use the JCR data control, which you can learn about in Chapter 6 “Integrating Content” in Oracle Fusion Middleware Developer's Guide for Oracle WebCenter.

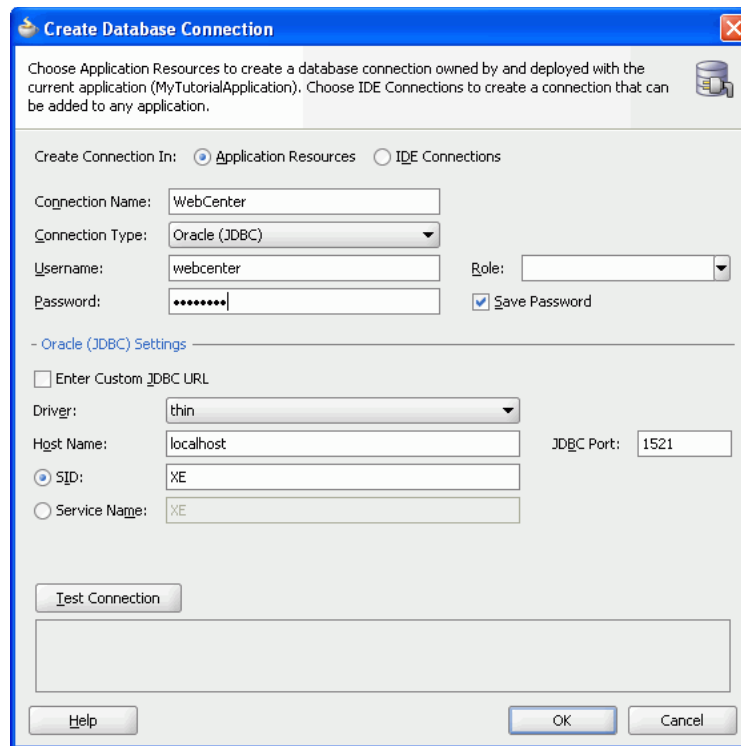
Step 5: Create a Database Connection to the WebCenter Schema for the Tags Service

Next, we'll add the Tags service. To use the Tags service, you must have access to the WebCenter schema. You installed this schema in [Chapter 2, "Preparing for the Tutorial."](#) In this section, we'll create a connection to the database containing this schema, so that the Tags service we add can use this schema.

To learn more about setting up your application for consuming services, refer to the Oracle Fusion Middleware Developer's Guide for Oracle WebCenter.

To create the database connection:

1. In the Application Navigator, expand the **Application Resources** panel.
2. Right-click **Connections**, choose **New Connection**, then choose **Database**.
3. Enter the following information for your database connection ([Figure 4–25](#)). Note that the Connection Name must be WebCenter.
 - **Connection Name:** WebCenter
 - **Connection Type:** Oracle (JDBC)
 - **User name:** username (for example, webcenter)
 - **Password:** password (for example, welcome1)
 - **Host:** <host where you installed the WebCenter schema> (for example, localhost)
 - **JDBC Port:** <port> (for example, 1521)
 - **SID:** <system identifier for the database with the same JDBC port> (for example, ORCL)

Figure 4–25 Database Connection

4. Click Test Connection to test your database connection. If you do not see a Success! message, check to make sure you entered the correct information for the WebCenter schema (see [Chapter 2, "Preparing for the Tutorial"](#) if you're not sure what you entered when you install it).
5. Click OK. Now that we've created our database connection, we can add the Tags service to the application.

Step 6: Add the Tags Service to Your Application

Tags enable you and your application users to apply your own meaningful terms to items in your application, making those items more easily discoverable in search results. Because your application *users* can create tags for their own content or content they've searched for, the tags are much more powerful and usable than search keywords created by application *developers* who may not be as familiar with the user-created content. This user-powered keyword creation makes the content in your application highly searchable and discoverable.

A "tag cloud" is a visual illustration of all the tags for the application, making it easy for you and your users to identify the tags used in the application. You can then search for a tag in your application to locate any item that has been associated with that tag.

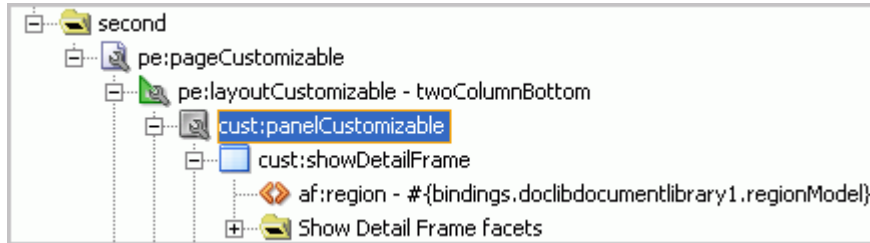
With every service in the Oracle WebCenter Framework, we perform a few basic steps: configure the service with our application, add the service task flow to our page, then run the page and customize or use the service at runtime.

In this section, we'll add the Tags service to our application and display it on our page. We will add a Tag Cloud so that we can visualize the tags and enable us to add new tags to our application.

To add the Tags service task flow:

1. In the Structure window for MyPage, navigate to the **Panel Customizable** containing the Show Detail Frame where you added the Document Library View (Figure 4-26).

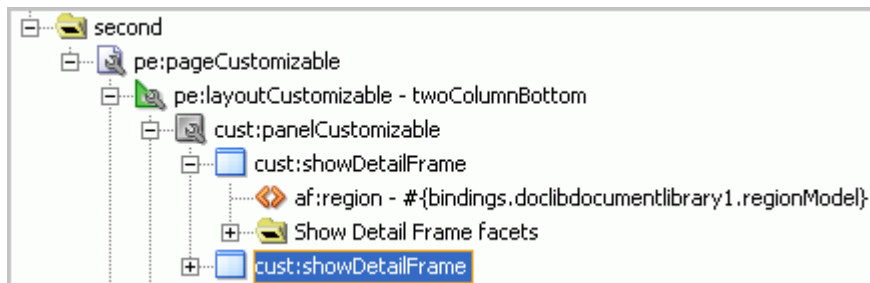
Figure 4-26 Panel Customizable Containing the Show Detail Frame



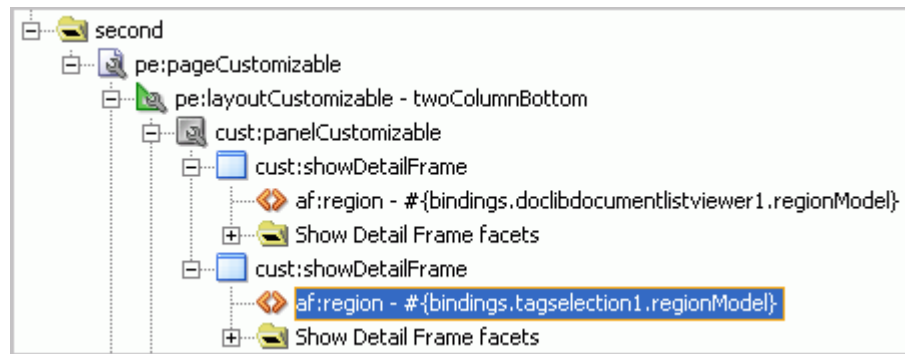
Note: As previously mentioned, you can use the pushpin in the Structure window to freeze the current view. For this step, you should click **MyPage** in the Design view, then ensure the pushpin is in the “freeze” position (pressed).

2. In the Component Palette, choose **Oracle Composer** from the list.
3. Under Layout, drag and drop a **Show Detail Frame** onto the Panel Customizable, so that it displays just below the first Show Detail Frame, as shown in Figure 4-27.

Figure 4-27 Second Show Detail Frame



4. While the Show Detail Frame is selected, in the Property Inspector, under Common, change the **Text** property to Tag Cloud.
5. In the Resource Palette, under WebCenter Service Catalog, open the **Task Flows** folder and locate the **Tagging - Tag Cloud** task flow, then drag and drop it onto the Show Detail Frame you just added.
6. In the Create Region context menu, choose **Create Region**. The Tagging - Tag Cloud task flow displays in the Structure window, as shown in Figure 4-28.

Figure 4–28 Tagging - Tag Cloud Task Flow in the Second Show Detail Frame

7. Right click MyPage in the Design view and choose **Run**. We'll examine the Tags service at runtime in the next step.

Step 7: Use, Add, and Search Tags in Your Application at Runtime

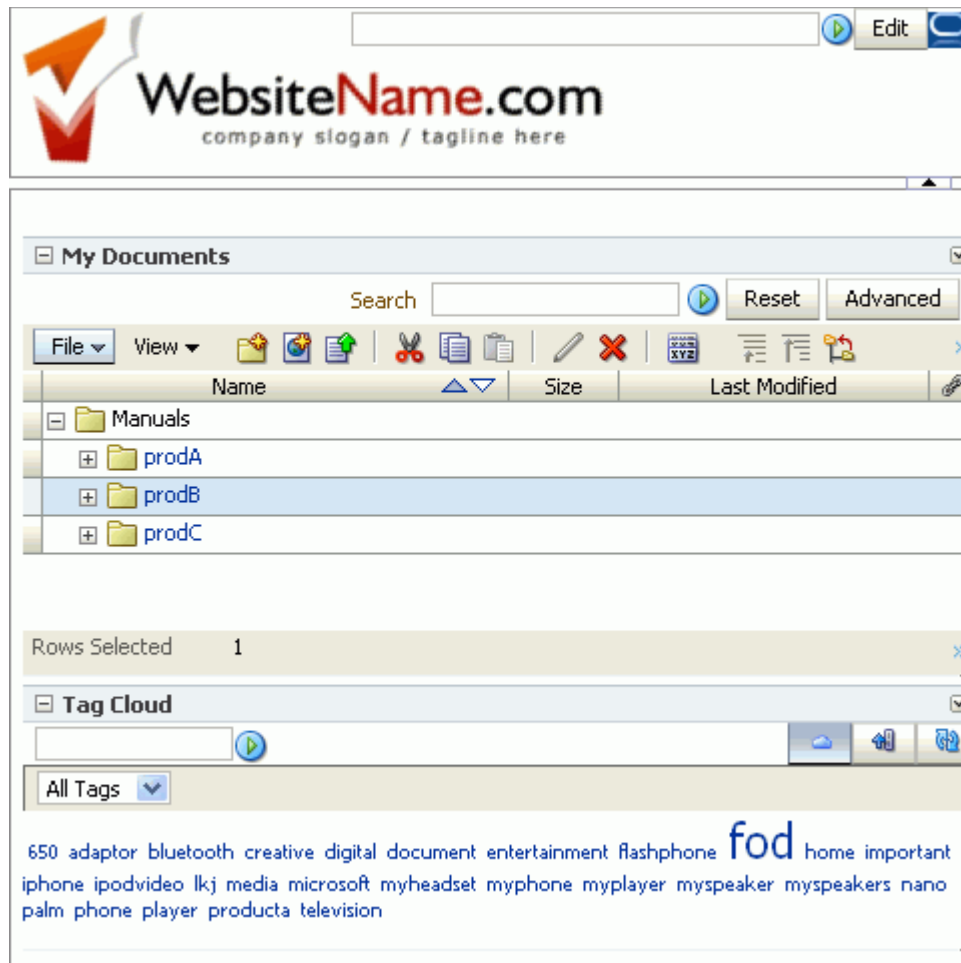
After you add the Tagging - Tag Cloud task flow to your page, the Tag Cloud automatically populates with the tags in the WebCenter. If you installed the WebCenter schema for the purposes of this tutorial, you will not see any tags in the Tag Cloud. In our example, we have a few tags that we populated to illustrate how the Tag Cloud can look at runtime.

To use the Tags service at runtime:

1. In your browser, you should now see the Tag Cloud displaying below the Document Library. In our example, we have a few sample tags that exist in our WebCenter schema. If a tag cloud exists in your WebCenter schema, the contents of your tag cloud may appear slightly different. If you do not have any tags, you will not see any tags in the Tag Cloud.

[Figure 4–29](#) shows how MyPage looks at runtime with sample tags in the Tag Cloud.

Figure 4–29 Tag Cloud at Runtime

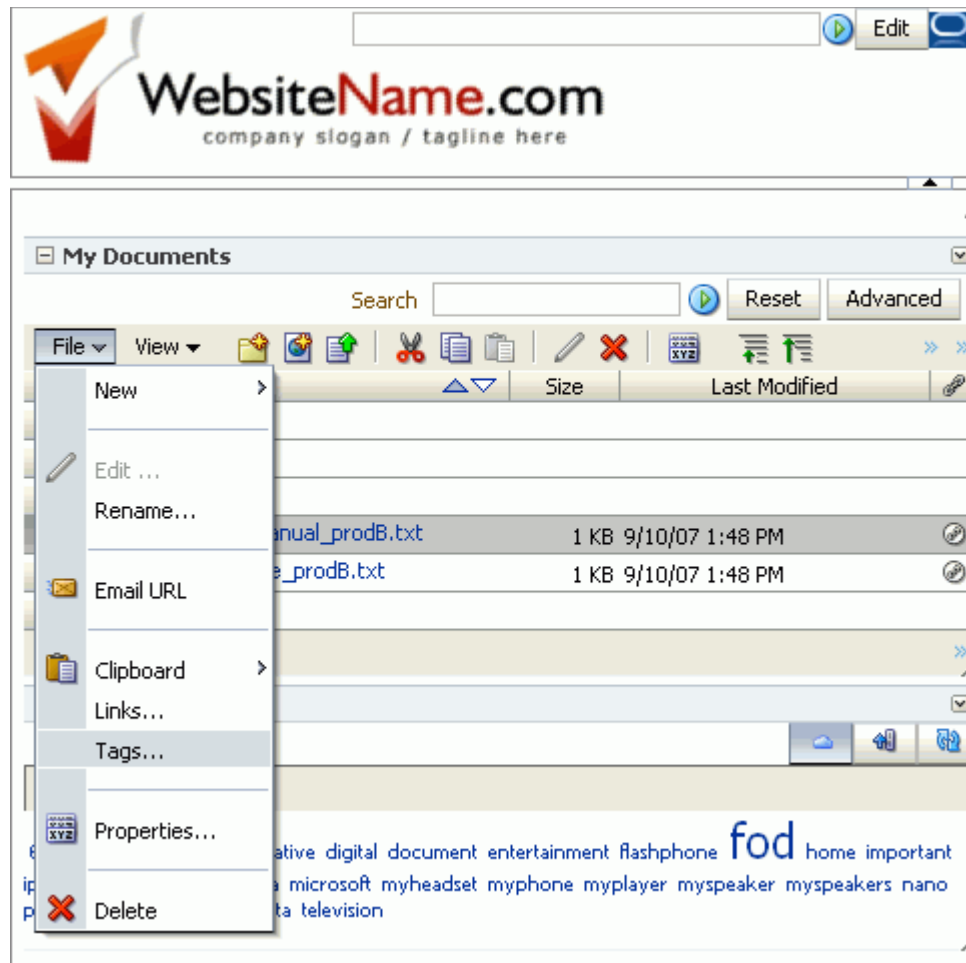


2. Once you add the Tags service to your application, you can use it with the Document Library service. Let's see how we can add tags to a document in our document library.

In the Document Library, open a folder, such as **prodB**, and select the row containing a document name, such as **productmanual_prodB.txt**. Be sure to click the row and not the link for the filename, otherwise you'll just display the text file in a new browser window as you did in [Step 4: Browse Documents at Runtime](#).

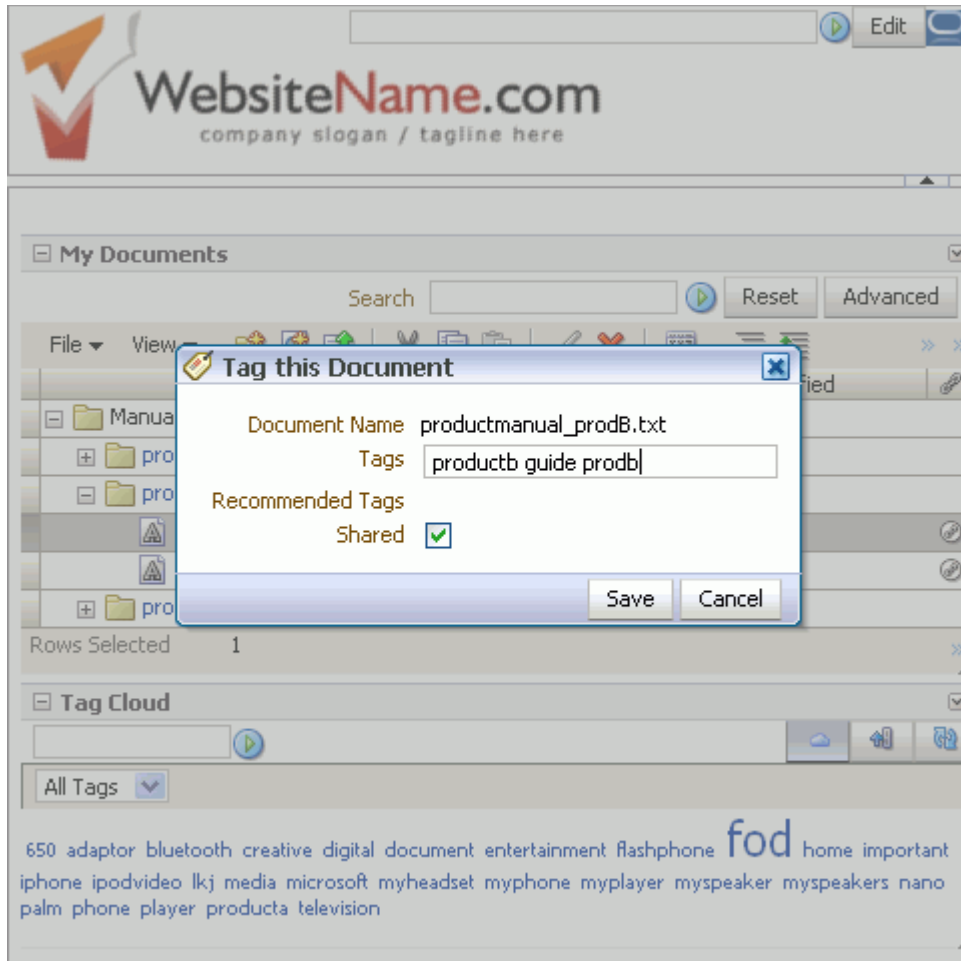
3. In the Document Library menu, choose **File > Tags...** ([Figure 4–30](#)).

Figure 4-30 Choosing the Tags Menu Option in the Document Library



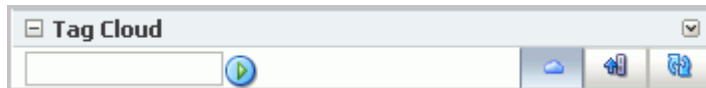
4. The Tag this Document dialog displays. Add a few tags, such as:
productb guide prodb

Figure 4–31 Adding New Tags to a Document

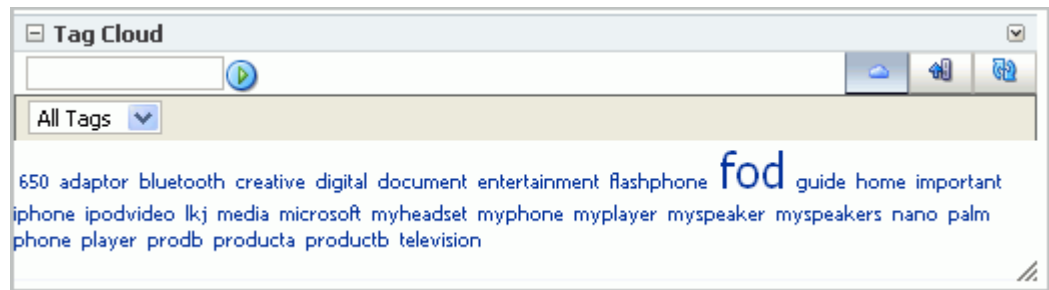


5. Click **Save**, then create another tag on another document. For example, open the **prodA** folder, then add a few tags to **productmanual_prodB.txt**.
6. In the Tag this Document dialog box, enter a few tags, then save your changes:
producta guide proda
7. Click the **Refresh** icon to the right of the Tag Cloud title bar (Figure 4–32).

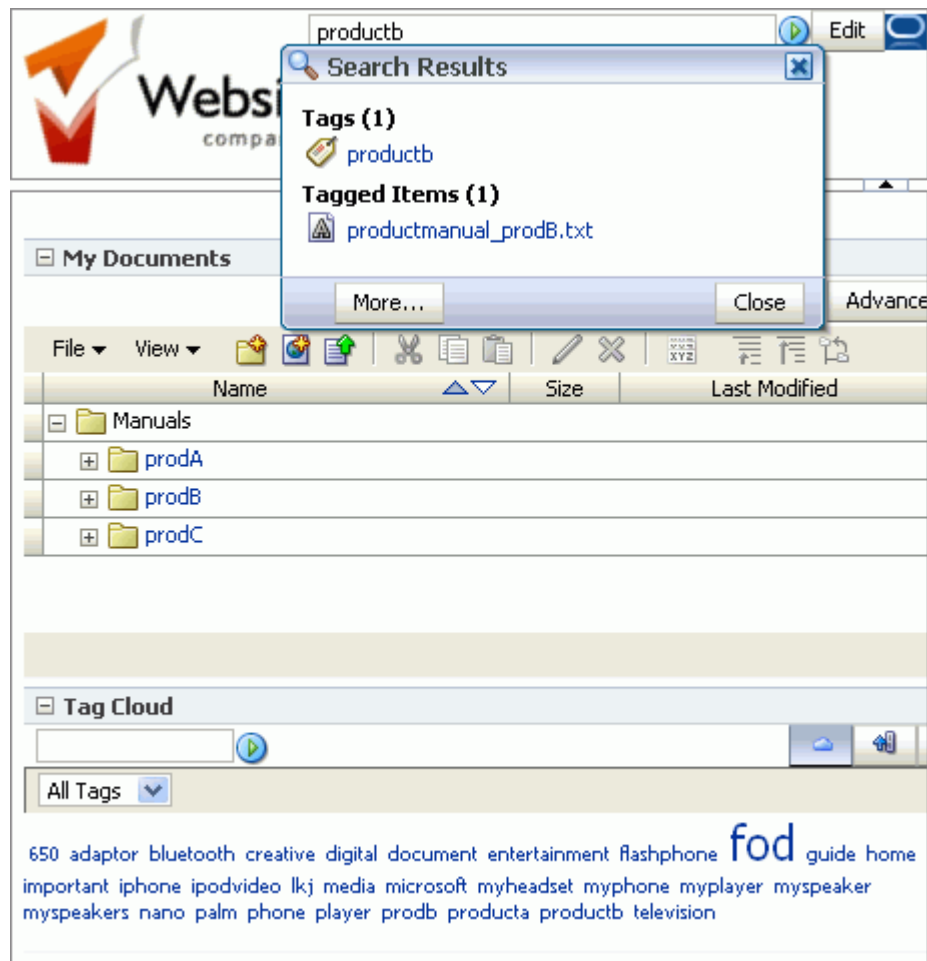
Figure 4–32 Refresh Tag Cloud Icon



Notice that the Tag Cloud now displays your new tags, for example **producta**, **productb**, **guide**, and **prodb** as shown in Figure 4–33.

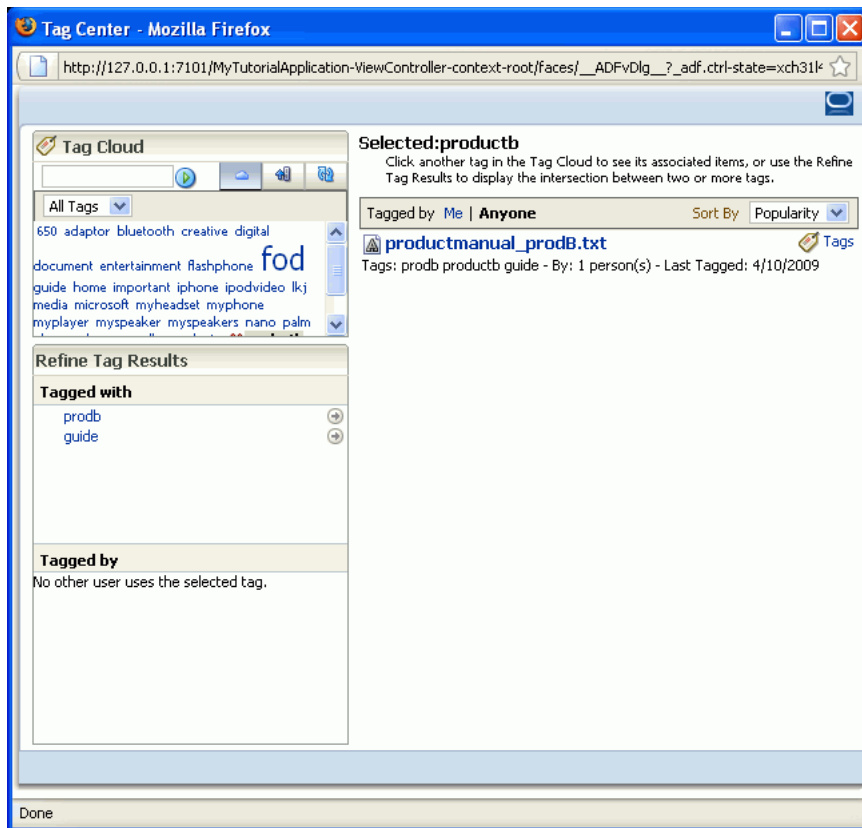
Figure 4–33 New Tags in the Tag Cloud

8. Let's see how tags help us locate items in our application. In the Search toolbar at the top of the page, enter `productb`, then click the arrow icon. You'll see the Search Results return the tag for `productb` (Figure 4–34).

Figure 4–34 productb Tag in the Search Results

9. In the Search Results window, click the **productb** tag. The Tag Center displays in a pop-up window (Figure 4–35).

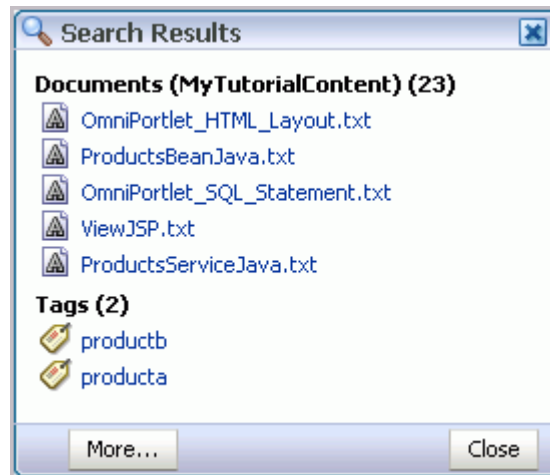
Figure 4–35 Tag Center



The Tag Center displays the Tag Cloud, all the results from your search that are associated with the tag `productb`, including other advanced search options. You can close this window after you're done viewing the different options and return to your application.

10. If you're interested, add a few more tags to see how the Tag Cloud changes. Otherwise, you can move on to the next step.
11. Let's examine the Search toolbar again. You can also search for any keyword to view items that contain the keyword. The keyword does not necessarily have to be a tag.

For example, in the Search toolbar, enter the keyword `product`, then click the arrow. The search returns all documents and tags that contain the word "product," as shown in [Figure 4–36](#).

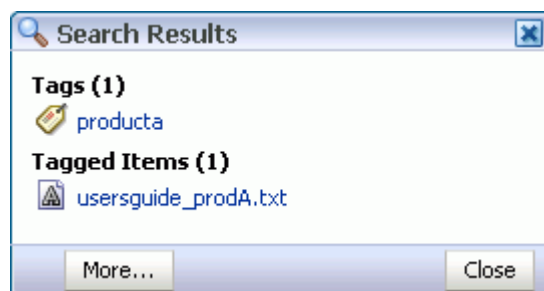
Figure 4–36 Search Results for “product”

Note that the search results in our example may not *exactly* reflect what you see in your application, as the results depend on the items you tagged.

12. When you search for a tag, the search returns a *partial* match -- that is, because we searched for `product`, we saw in our Search Results tags and documents that contain the word `product`. But, because no item was specifically tagged with “product,” the search did not return any tagged items.

Let’s search for a specific tag and see what happens. In the Search toolbar, enter `producta`, then click the arrow icon.

[Figure 4–37](#) shows the search results. Notice that the *tagged item*, `usersguide_prodB.txt` displays. Because the item was tagged with “producta,” our search returned the specific item.

Figure 4–37 Search Results for “producta”

You can learn more about adding the Search, Documents, and Tags services to a custom WebCenter application in Oracle Fusion Middleware Developer’s Guide for Oracle WebCenter. You can learn more about using these services at runtime (in your browser) in the Oracle Fusion Middleware User’s Guide for Oracle WebCenter.

Now that we have added a few services to our application, let’s build a few portlets, add them to the application, then wire them in [Chapter 5, “Building Portlets and Wiring Them in Your Application.”](#)

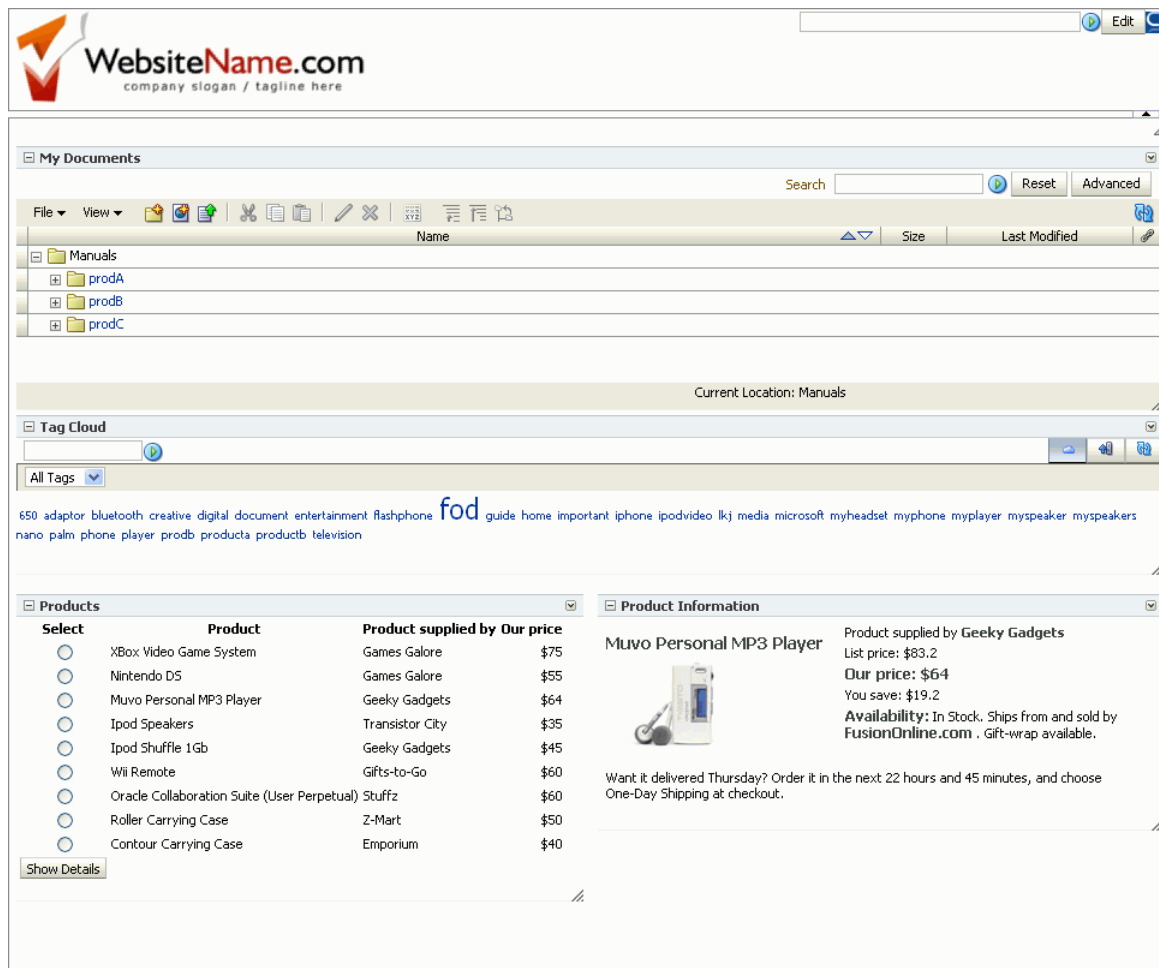
Building Portlets and Wiring Them in Your Application

In this lesson, you will learn how to build two types of portlets: a standards-based Java (JSR 168) portlet, which you will build using a wizard in Oracle JDeveloper, and an OmniPortlet, using a step-by-step wizard at runtime.

After you create the portlets, you will add them to the page, then connect the two portlets. By the end of this lesson, you should have a good handle on what's involved with building and testing a standards-based Java (JSR 168) portlet and a PDK-Java portlet (OmniPortlet). You'll also be able to "wire" the two portlets so that when you click a link in one portlet, the content in the second portlet is dynamically updated.

[Figure 5-1](#) shows how your page will look at the end of this lesson.

Figure 5–1 MyPage.jspx at the End of this Lesson



Introduction

This lesson contains the following steps:

- Step 1: Create a Standards-Based Java (JSR 168) Portlet
- Step 2: Create the JavaBeans to Store the Standards-Based Portlet Information
- Step 3: Create the Business Logic for the Standards-Based Portlet
- Step 4: Test and Deploy the Standards-Based Portlet
- Step 5: Register the Standards-Based Portlet with Your Application
- Step 6: Test the Standards-Based Portlet in Your Application
- Step 7: Register the Preconfigured Portlet Producer
- Step 8: Add an OmniPortlet to Your Page
- Step 9: Define OmniPortlet at Runtime
- Step 10: Wire the Standards-Based Portlet and OmniPortlet Together
- Step 11: Test the Interaction Between the Portlets

Both of these portlets use the tutorial schema that we installed in [Chapter 2, "Preparing for the Tutorial,"](#) and require that you have a database connection. We created the database connection when we added the Tags service in [Chapter 4, "Adding WebCenter Web 2.0 Services to Your Application,"](#) so if you did not complete the steps in that chapter, you must follow the steps in [Step 5: Create a Database Connection to the WebCenter Schema for the Tags Service](#) in that chapter before you build the portlets.

Step 1: Create a Standards-Based Java (JSR 168) Portlet

Oracle WebCenter Framework enables you to quickly and easily build a standards-based portlet that you can use with a portal or application, such as the one you're currently creating.

Note: The steps to build this portlet rely on the naming convention we've used, so follow the steps carefully. If you do not use the names we've provided, you may not achieve the same results.

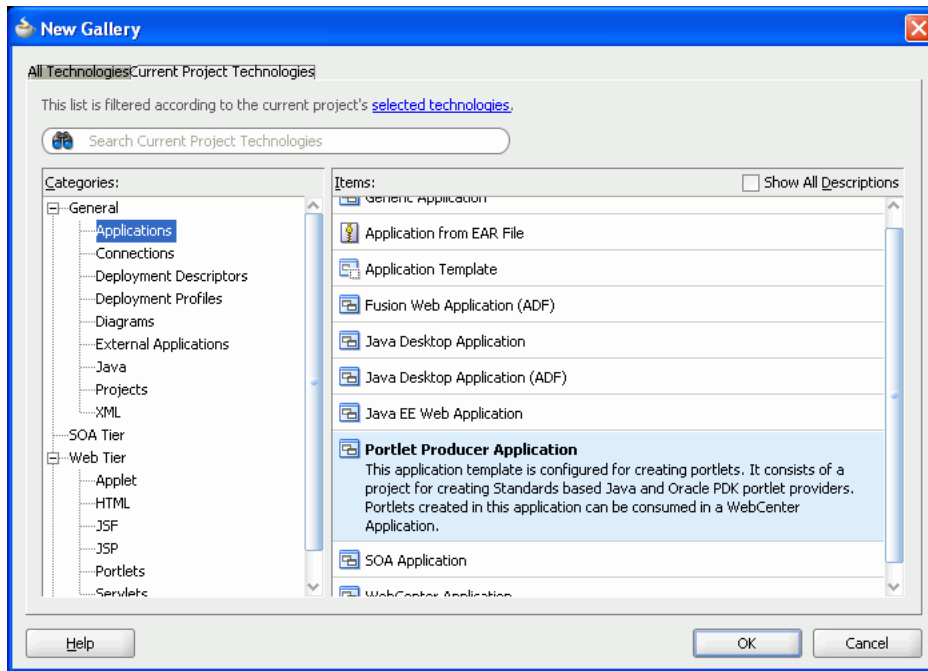
In this step, we will create an application based on the Portlet Producer template, then build a standards-based portlet. Afterwards, we'll consume the portlet into our tutorial application. [Figure 5-2](#) shows the portlet at runtime in `MyTutorialApplication`.

Figure 5-2 Standards-Based Portlet at Runtime in MyTutorialApplication

Select	Product	Product supplied by	Our price
<input type="radio"/>	XBox Video Game System	Games Galore	\$75
<input type="radio"/>	Nintendo DS	Games Galore	\$55
<input type="radio"/>	Muvo Personal MP3 Player	Geeky Gadgets	\$64
<input type="radio"/>	Ipod Speakers	Transistor City	\$35
<input type="radio"/>	Ipod Shuffle 1Gb	Geeky Gadgets	\$45
<input type="radio"/>	Wii Remote	Gifts-to-Go	\$60
<input type="radio"/>	Oracle Collaboration Suite (User Perpetual)	Stuffz	\$60
<input type="radio"/>	Roller Carrying Case	Z-Mart	\$50
<input type="radio"/>	Contour Carrying Case	Emporium	\$40

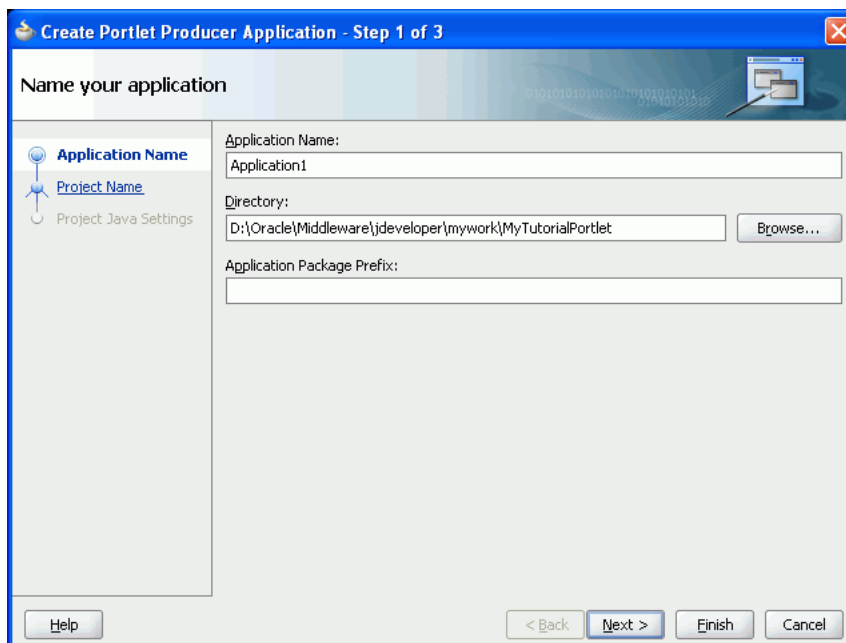
1. In Oracle JDeveloper, create a new application based on the **Portlet Producer Application** template ([Figure 5-3](#)), then click **OK**. To do so, choose **File > New**, then choose **Applications > Portlet Producer Application** in the New Gallery.

Figure 5-3 Creating a New Portlet Producer Application



2. In the Create Portlet Producer Application wizard, in the Application Name field, enter MyTutorialPortlet.
3. By default, the Directory field should contain the directory where the application will reside (Figure 5-4). You can change the directory location, if you like, but let's leave it as it is for the purposes of the tutorial.

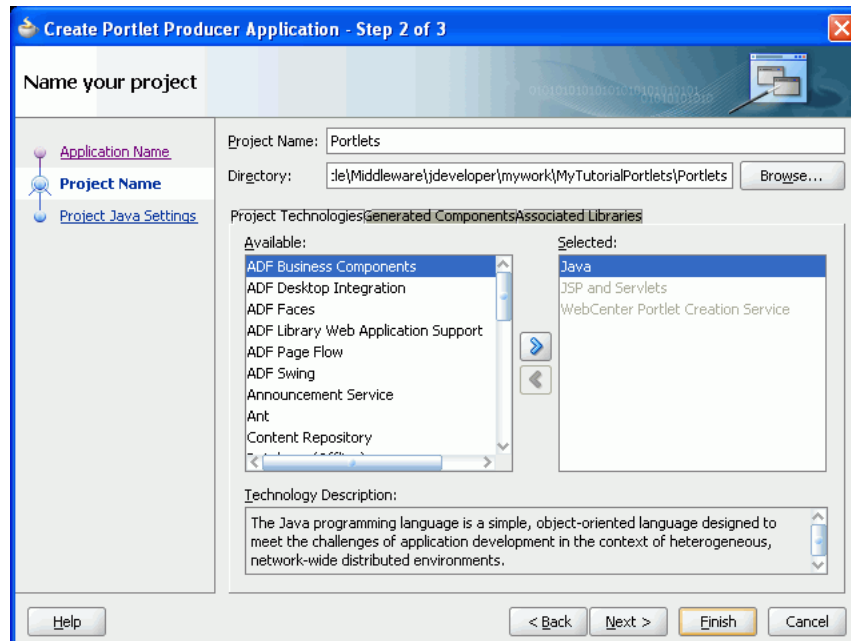
Figure 5-4 Creating a Portlet Producer Application - Step 1



4. Click Next.

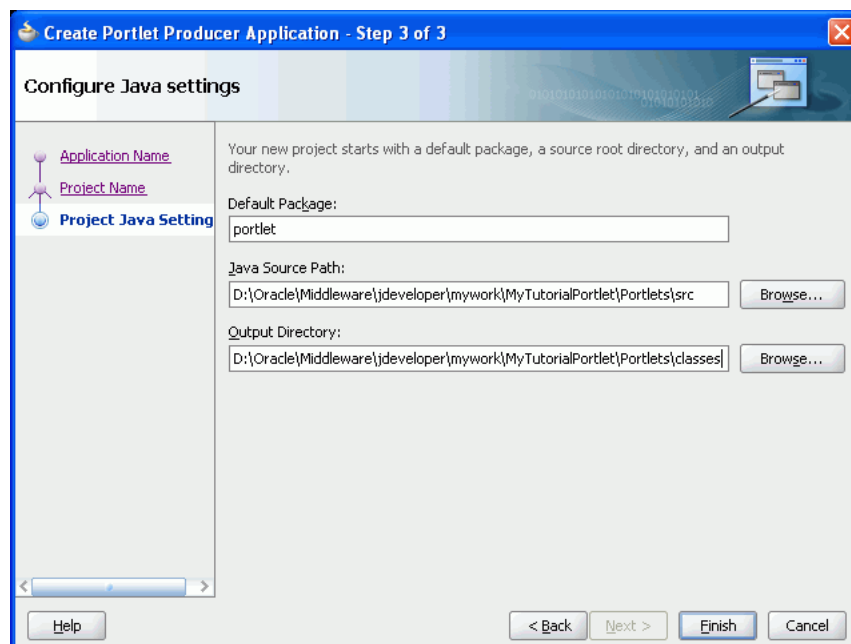
- On the Project Name page of the wizard (Figure 5-5), let's leave the default name: Portlets, and click Next.

Figure 5-5 Creating a Portlet Producer Application - Step 2



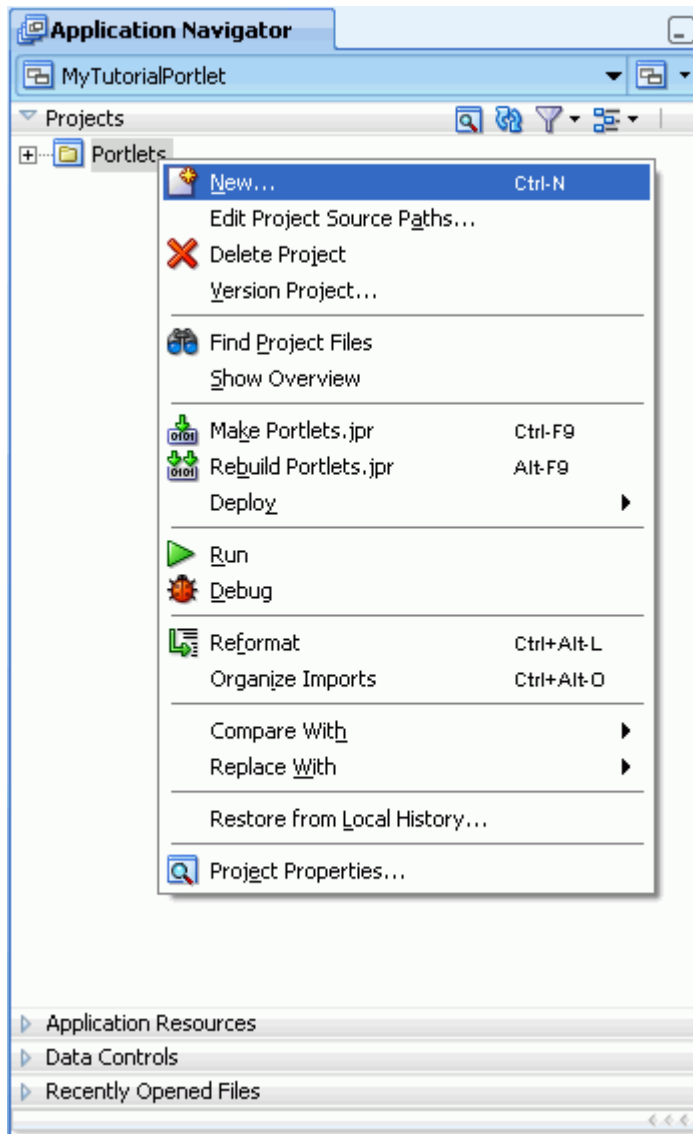
- On the last page of the wizard, you can configure the Java settings. You can see that the default package contained in the portlet producer application is `portlet` (Figure 5-6). For the purposes of this tutorial, let's leave the default options and click Finish.

Figure 5-6 Creating a Portlet Producer Application - Step 3

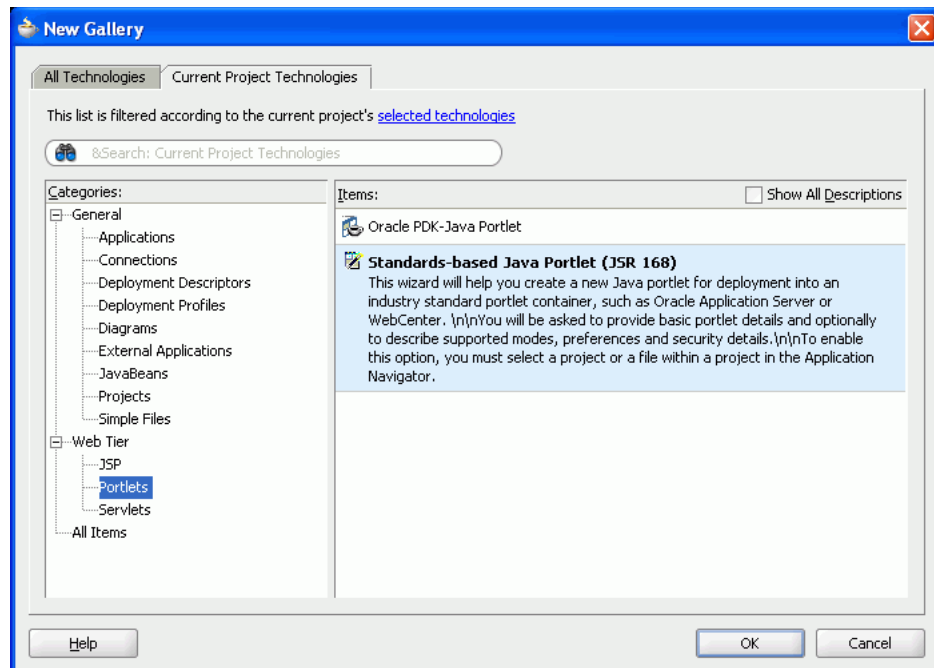


7. In the Application Navigator, under your new MyTutorialPortlet application, right-click **Portlets**, and select **New** (Figure 5-7).

Figure 5-7 Selecting New from the Menu



8. Click the **Current Project Technologies** tab.
9. In the Categories list as shown in Figure 5-8, expand the **Web Tier** category, and select **Portlets**.

Figure 5–8 Creating a New Portlet Producer Application

Notice there are two kinds of portlets you can create:

- An Oracle PDK-Java portlet. PDK-Java portlets can be consumed by WebCenter applications, Oracle Portal, or some other type of Oracle-specific solution. You build an Oracle PDK-Java portlet using the APIs provided by the PDK. Note that the OmniPortlet producer you will register in [Step 7: Register the Preconfigured Portlet Producer](#) is a type of Oracle PDK-Java Portlet
- A standards-based Java (JSR 168) Java portlet. Java portlets can be consumed by portals from any vendor that supports the portlet standards. In this tutorial, we're going to build a standards-based (JSR 168) Java portlet.

10. Select **Standards-based Java Portlet (JSR 168), and click **OK**.**

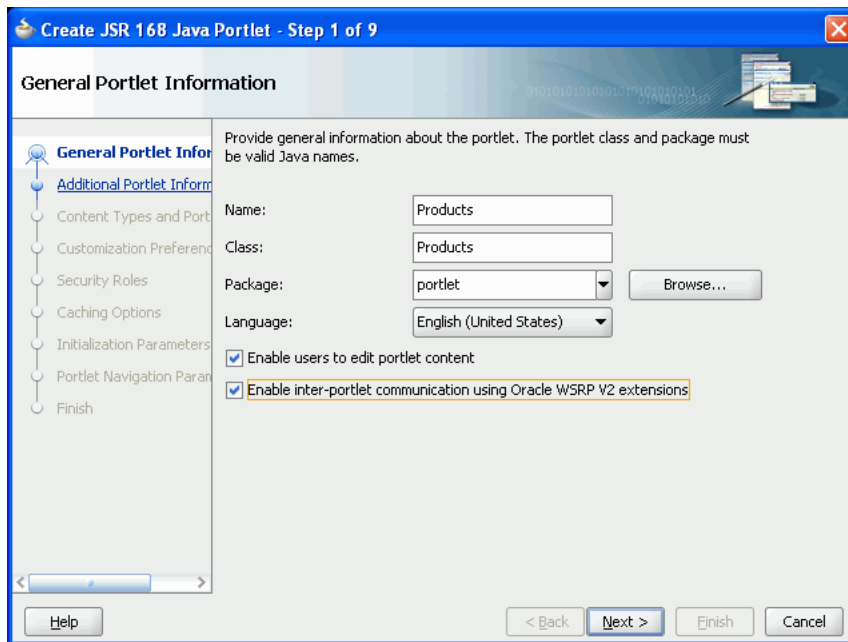
The JSR 168 Java Portlet wizard displays, which generates a skeleton for the portlet. We'll later add our own logic to the portlet. Let's see how this is done.

11. On the General Portlet Information page, in the Name and Class fields, enter `Products`.

Note: The steps to build this portlet rely on the naming convention we've used, so follow the steps carefully. If you do not use the names we've provided, you may not achieve the same results.

- 12. Select **Enable inter-portlet communication using Oracle WSRP V2 extensions**.** Selecting this option enables your portlet to support Oracle WSRP 2.0 extensions, and generates the `oracle-portlet.xml` file that is used for WSRP 2.0 features, such as navigation parameters. We'll need these parameters later on, when we enable the OmniPortlet and this portlet to communicate with each other. [Figure 5–9](#) shows how the General Portlet Information page should now look.

Figure 5–9 Creating a JSR 168 Java Portlet - General Portlet Information



13. Click **Next**.

14. On the Additional Portlet Information page, we can either leave the defaults or, because we know we are going to show a few details about the products in our database, we can change the display name so anyone using the portlet will know what the portlet contains. Update the fields on this page according to [Table 5–1](#). [Figure 5–10](#) shows the resulting Additional Portlet Information page.

Table 5–1 Name and Attribution Values

Property	Value
Display Name	Name that will appear in the JDeveloper Component Palette. Because you entered Products as the class name, this field is automatically populated with that name.
Portlet Title	Title that will appear on the portlet header. Because you entered Products as the class name, this field is automatically populated with that name.
Short Title	Title that will appear on the portlet header on mobile devices. Let's leave the default name, Products.
Description	Description of the portlet. This field is relevant only when the portlet is used in an Oracle Portal 10g environment. Enter a description, for example, This is a JSR 168 portlet that displays the products.
Keywords	Keywords provide additional information about a page, item, or portlet so that users can locate it during a search. Although keywords are not supported by Oracle WebCenter Suite or Oracle Portal 10g, they are supported by other vendors from whom you may have obtained a deployment environment. Enter sample, Tutorial, products.

Figure 5–10 Creating a JSR 168 Java Portlet - Additional Portlet Information

Create JSR 168 Java Portlet - Step 2 of 9

Additional Portlet Information

Provide additional information about the portlet. Meaningful display names, descriptions, and keywords help users find the portlets they want.

General Portlet Information
 Additional Portlet Information
 Content Types and Portlet Modes
 Customization Preferences
 Security Roles
 Caching Options
 Initialization Parameters
 Portlet Navigation Parameters
 Finish

Display Name:
 Identifies the portlet to Oracle Portal users.

Portlet Title:
 Displayed in the portlet header.

Short Title:
 Used for clients with limited display space.

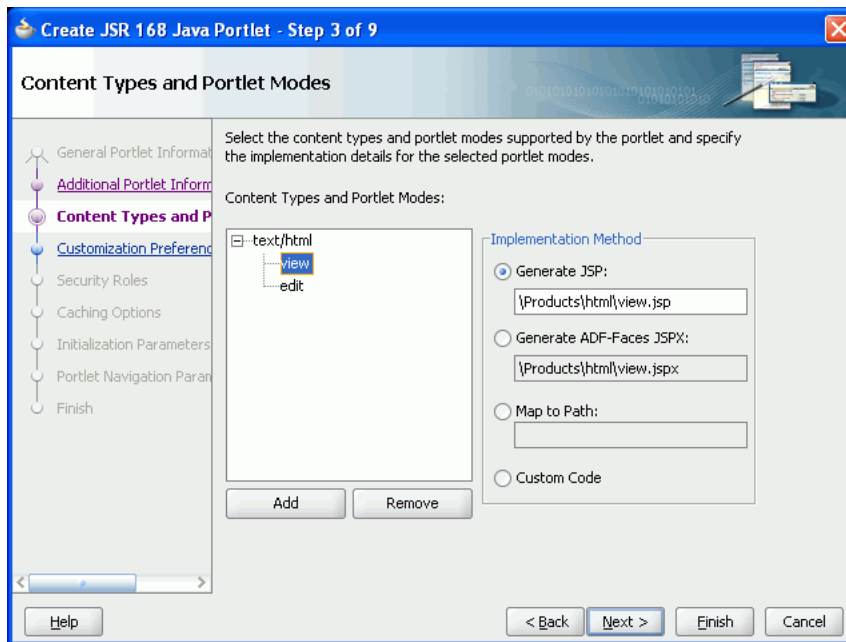
Description:

Keywords:
 Separate multiple entries with commas.

15. Click **Next**.

16. On the Content Types and Portlet Modes page, notice that **text/html** is the default content type. That means that the portlet will support text encoded with HTML. View and edit are listed as the default portlet modes for `text/html`. View is always available as a portlet mode; edit mode provides a page that allows users to personalize the portlet instance.

Notice the Implementation Method area as shown in [Figure 5–11](#). These controls enable you to specify whether you want to generate JSP for the portlet, or use your own custom JSP code.

Figure 5–11 Creating a JSR 168 Portlet - Content Types and Portlet Modes

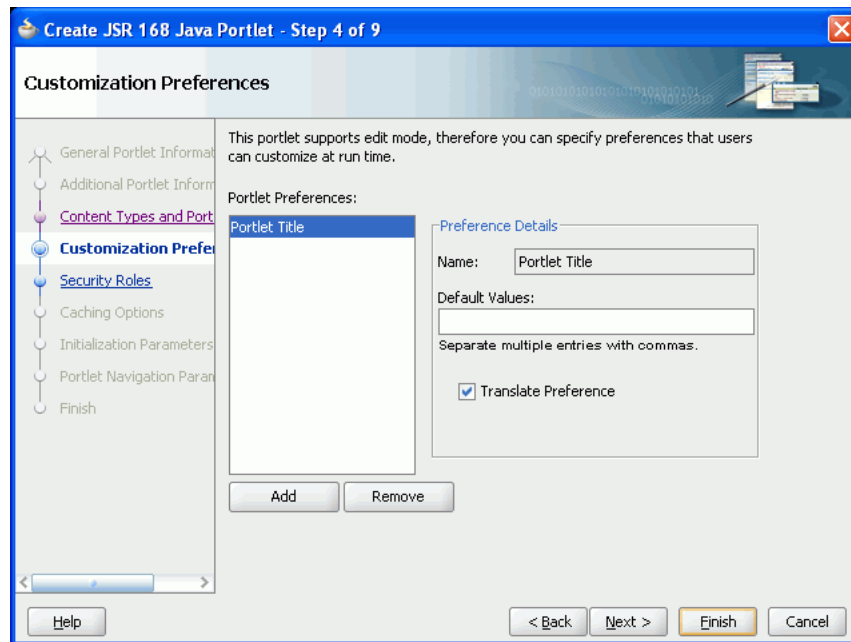
In this lesson, we'll ask JDeveloper to generate JSPs for us by leaving the default selection.

17. Click *Next*.

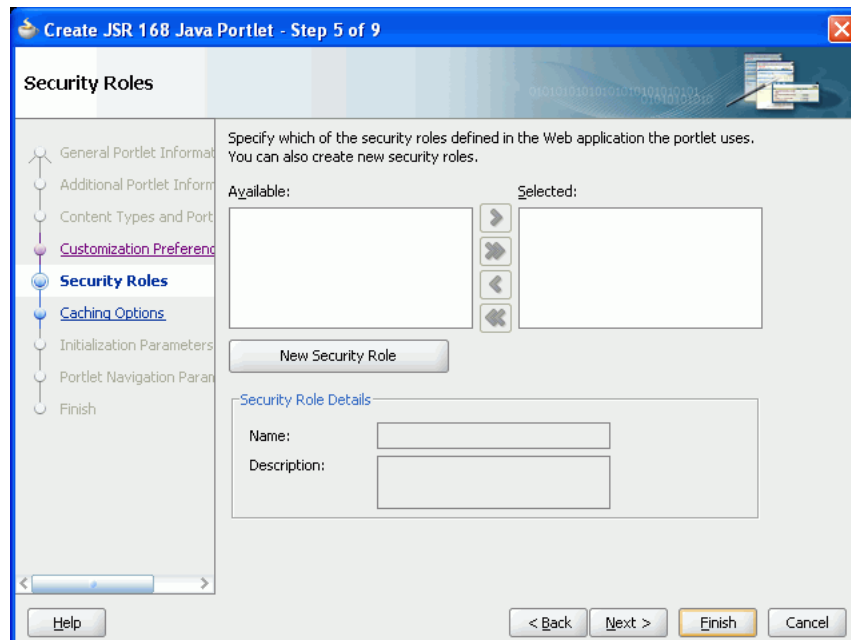
Although you could click **Finish** here and produce a basic portlet, let's continue and choose some other options and settings for our portlet.

18. On the Customization Preferences page, let's leave the default values and click *Next* (Figure 5–12).

Although we're not going to do anything with this page now, in the future you can use it to add other customization options for the portlet. For example, if your portlet accepted a `zip` parameter, you might want to allow users to personalize the Zip Code label. If this were the case, you would use the **Add** button to make the Zip Code label personalizable.

Figure 5–12 *Creating a JSR 168 Java Portlet - Customization Preferences*

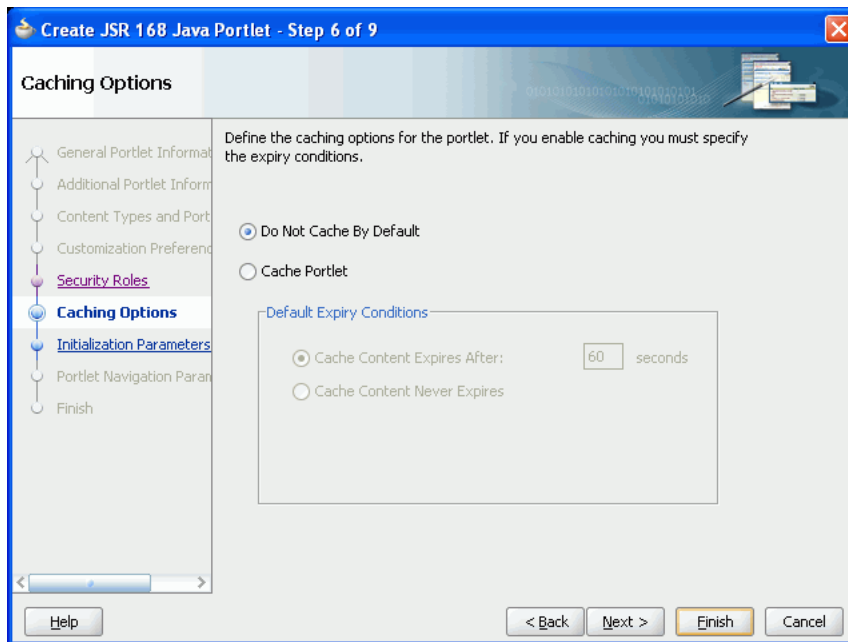
19. On the Security Roles page, click **Next** (Figure 5–13). The Security Roles page is used to specify which of the application's security roles you want to establish for this portlet.

Figure 5–13 *Creating a JSR 168 Java Portlet - Security Roles*

20. On the Caching Options page, click **Next** (Figure 5–14).

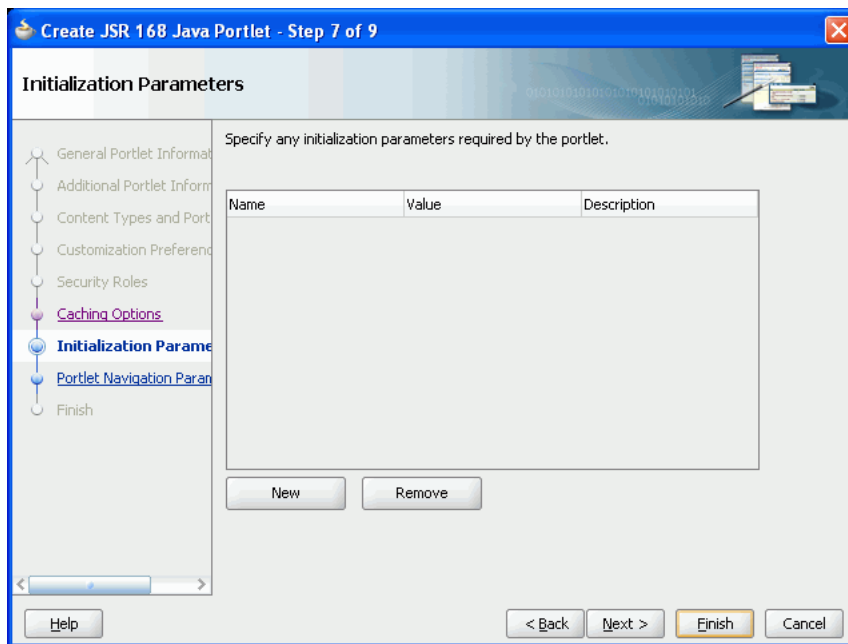
The settings on this page enable you to define expiry-based caching for your portlet. You do not need any caching conditions now.

Figure 5–14 *Creating a JSR 168 Java Portlet - Caching Options*



21. On the Initialization Parameters page, click **Next** (Figure 5–15). Our portlet does not require any initialization parameters.

Figure 5–15 *Creating a JSR 168 Java Portlet - Initialization Parameters*



22. On the Portlet Navigation Parameters page, let's create a navigation parameter based on the `product ID`.

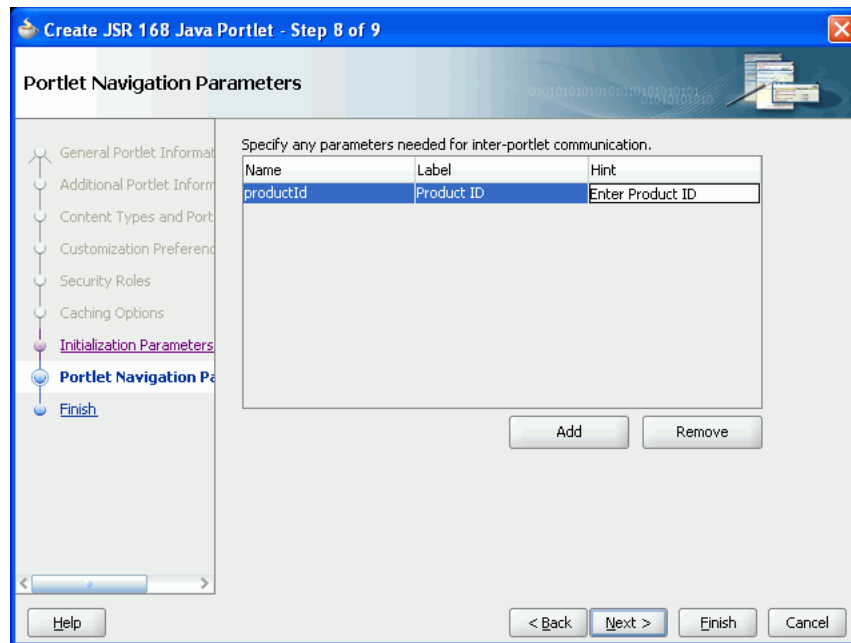
Navigation parameters are a WSRP 2.0 feature. This page enables you to specify external parameters to be consumed by the standards-based portlet, and only displays if you select the **Enable inter-portlet communication using Oracle WSRP V2 extensions** option on the first page of the wizard.

Click **Add**.

23. Double-click the values and update each value to the following:
 - **Name:** productId
 - **Label:** Product ID
 - **Hint:** Enter Product ID

We will use these navigation parameters later on in [Step 10: Wire the Standards-Based Portlet and OmniPortlet Together](#). [Figure 5–16](#) shows the updated Portlet Navigation Parameters page.

Figure 5–16 *Creating a JSR 168 Java Portlet - Portlet Navigation Parameters*

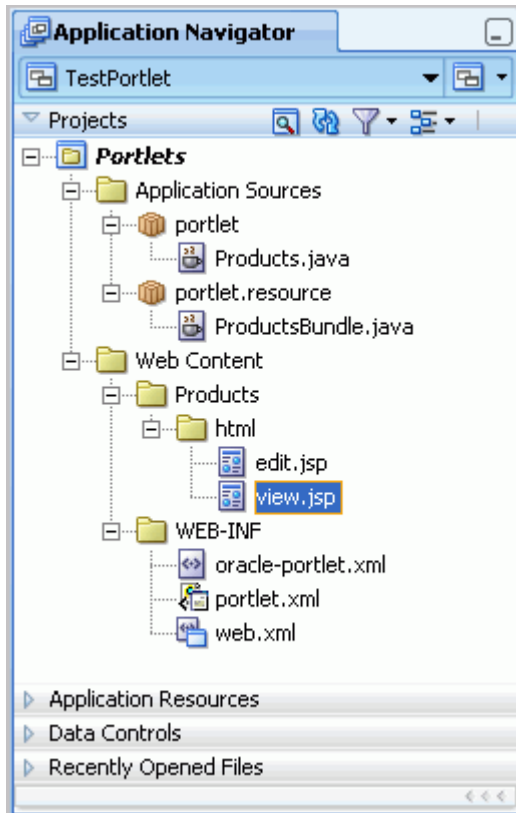


24. Click **Next**.

25. On the last page of the wizard (Step 9 of 9), click **Finish**.

After you click **Finish**, you should be able to locate several newly generated files in the Application Navigator under the **Portlets** project. The expanded Navigator looks like [Figure 5–17](#).

Figure 5–17 Files Generated for the New Portlet



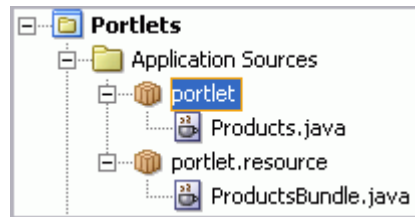
- Under Application Sources, under **portlet** and **portlet.resource**, notice two Java classes:
 - **Products.java** is invoked by the portlet container. It contains all the methods required by the portlet standards.
 - **ProductsBundle.java** contains all the translation strings for the portlet.
 - Under **Web Content, Products, html**:
 - **edit.jsp**, which contains the information needed to populate the Personalize dialog box.
 - **view.jsp**, which is invoked when the portlet is sharing the page with other components.
 - Under **Web Content, WEB-INF**, three deployment descriptors:
 - **oracle-portlet.xml**, which contains information to support Oracle extensions for import/export and inter-portlet communication. It appears because you chose **Enable WSRP V2 inter-portlet communication using Oracle extensions** on Step 1 of the wizard.
 - **portlet.xml**, which specifies all the portlet resources (the information you entered through the JSR 168 Java Portlet Wizard).
 - **web.xml**, which specifies the web application resources.
26. Save all your files. In the next step, we'll create a JavaBean to store all the portlet information you just generated.

Step 2: Create the JavaBeans to Store the Standards-Based Portlet Information

In this step, you will create the JavaBean to store the information for your standards-based portlet.

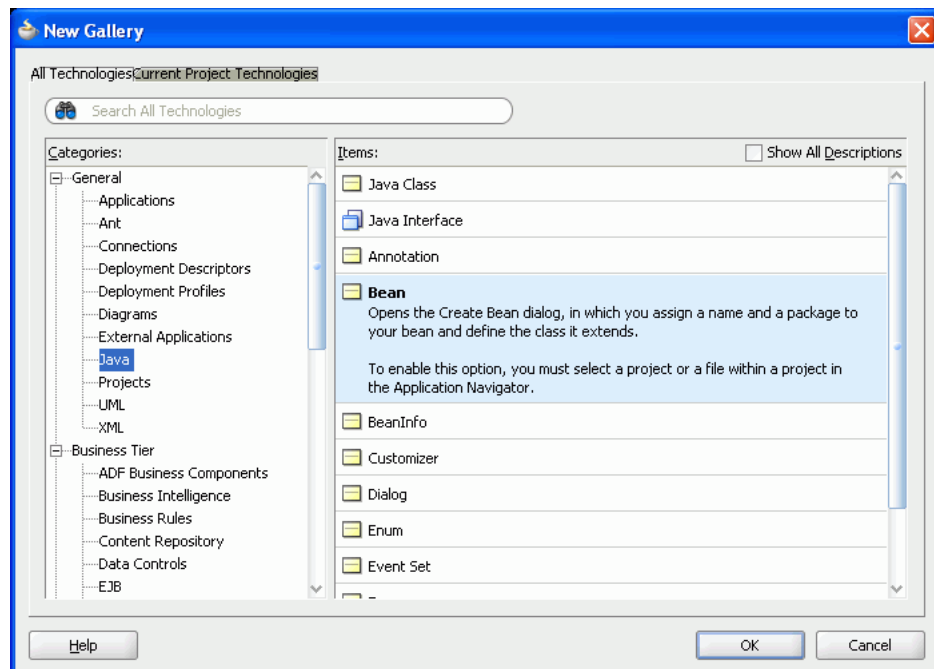
1. In the Application Navigator, right-click the **portlet** package, and choose **New**. [Figure 5-18](#) shows the `portlet` package in the Application Navigator.

Figure 5-18 Portlet Package



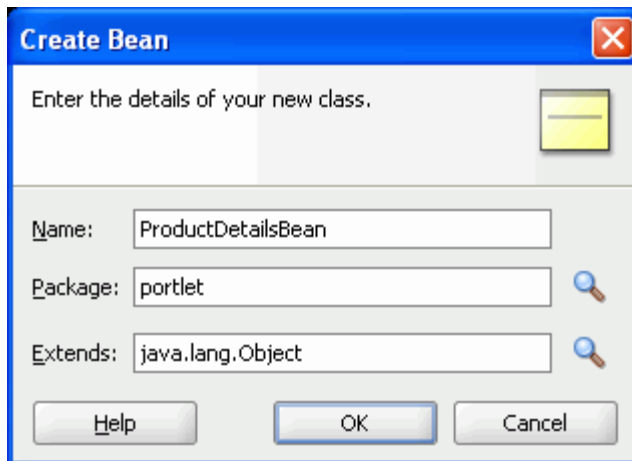
2. In the New Gallery, click the **All Technologies** tab.
3. In the Categories list, under **General**, choose **Java**, select **Bean** from the Items list, then click **OK** ([Figure 5-19](#)).

Figure 5-19 Choosing the JavaBean in the New Gallery



4. In the Create Bean dialog box that displays, enter the following information to set up the new JavaBean ([Figure 5-20](#)):
 - Name: `ProductDetailsBean`
 - Package: `portlet`
 - Extends: `java.lang.Object`

Figure 5–20 Create ProductDetailsBean



This creates a new bean called `ProductDetailsBean` in the `portlet` package.

5. Click **OK**. The new JavaBean displays in the Design window (Figure 5–21).

Figure 5–21 ProductDetails JavaBean in the Design View

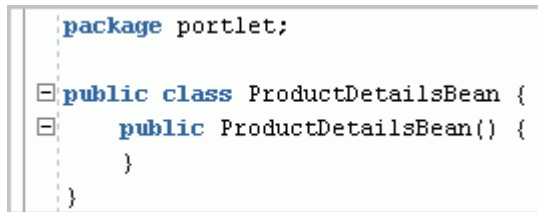
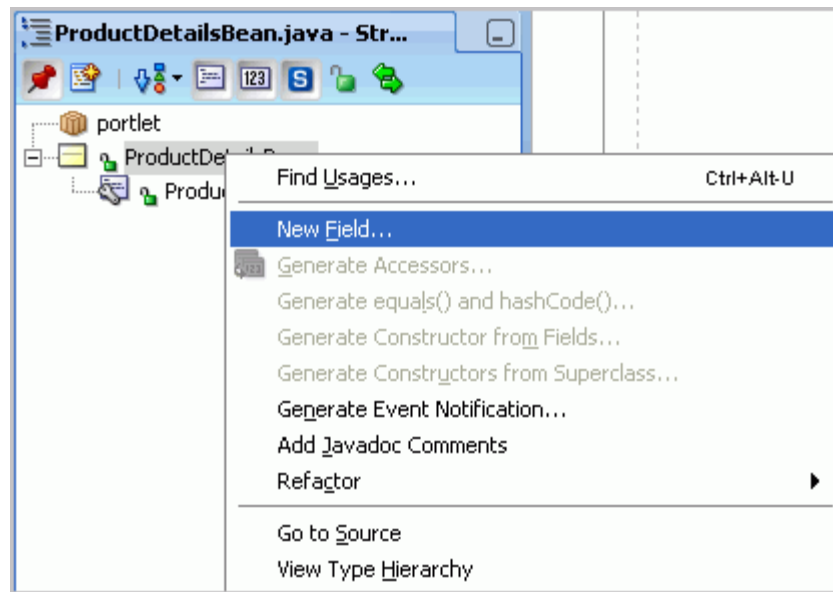


Figure 5–22 shows the JavaBean in the Structure window. As we mentioned in Chapter 3, "Creating a WebCenter Application with a Customizable Page," you can use the pushpin in the Structure window to freeze and unfreeze the current view. Ensure that you have selected the `ProductDetailsBean` in the Design view, then toggle the pushpin so that you see the `ProductDetailsBean` in the Structure window.

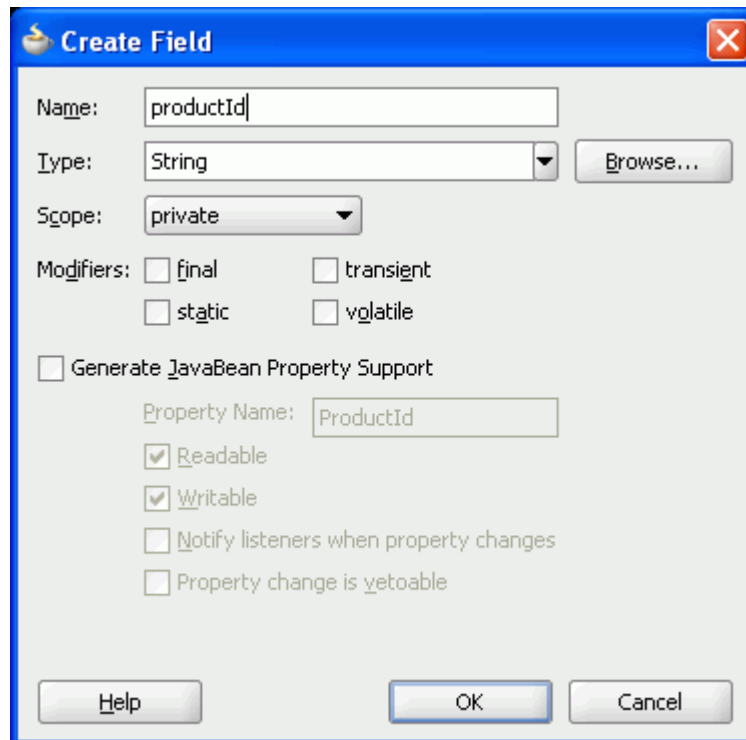
Figure 5–22 ProductDetailsBean in the Structure Window



6. Now that we've set up our JavaBean, let's add the information we need for the portlet. In the Structure window, right-click the **ProductDetailsBean**, then choose **New Field** from the context menu.

Figure 5–23 *Creating a New Field for the ProductDetails Bean*

7. In the Create Field dialog box, in the **Name** field, enter `productId`. This name represents the name of the product in our database schema.
8. Ensure the Type is set to `String`.
9. Ensure the Scope is set to **private** (Figure 5–24).

Figure 5–24 *Create Field Dialog Box*

10. Click **OK**. The new field displays in the Design view of the JavaBean (Figure 5–25).

Figure 5–25 New Field in the ProductDetailsBean

```
package portlet;

public class ProductDetailsBean {
    private String productId;

    public ProductDetailsBean() {
    }
}
```

11. Now, let's create the other four fields we want to show in our portlet. Follow steps 6 through 10 to create these four fields with the following names:

- productName
- productPrice
- imageURL
- categoryDescription
- supplierName

The Design view of your JavaBean should now contain the six fields ([Figure 5–26](#)).

Figure 5–26 ProductsBean with the Six Fields

```
package portlet;

public class ProductDetailsBean {
    private String productId;
    private String productName;
    private String productPrice;
    private String imageURL;
    private String categoryDescription;
    private String supplierName;

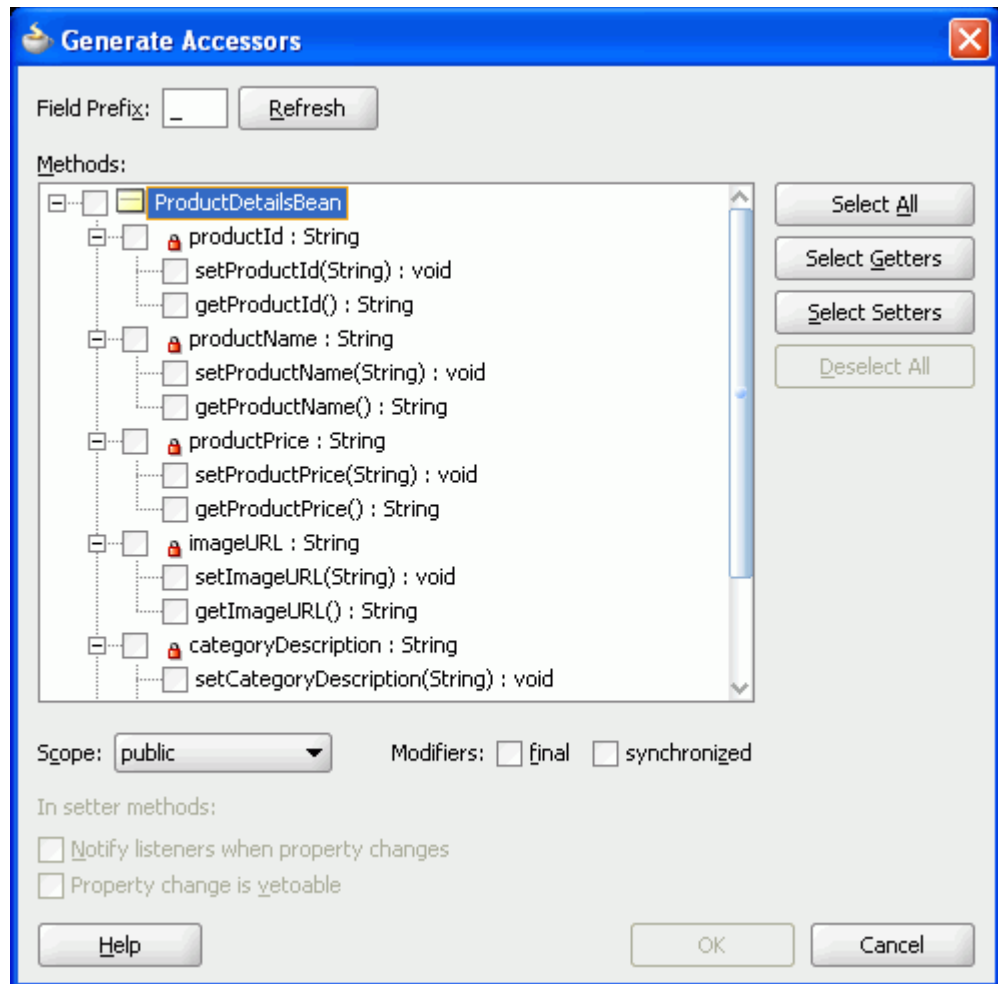
    public ProductDetailsBean() {
    }
}
```

12. Now that we've set up the fields for the JavaBean, let's generate the accessors.

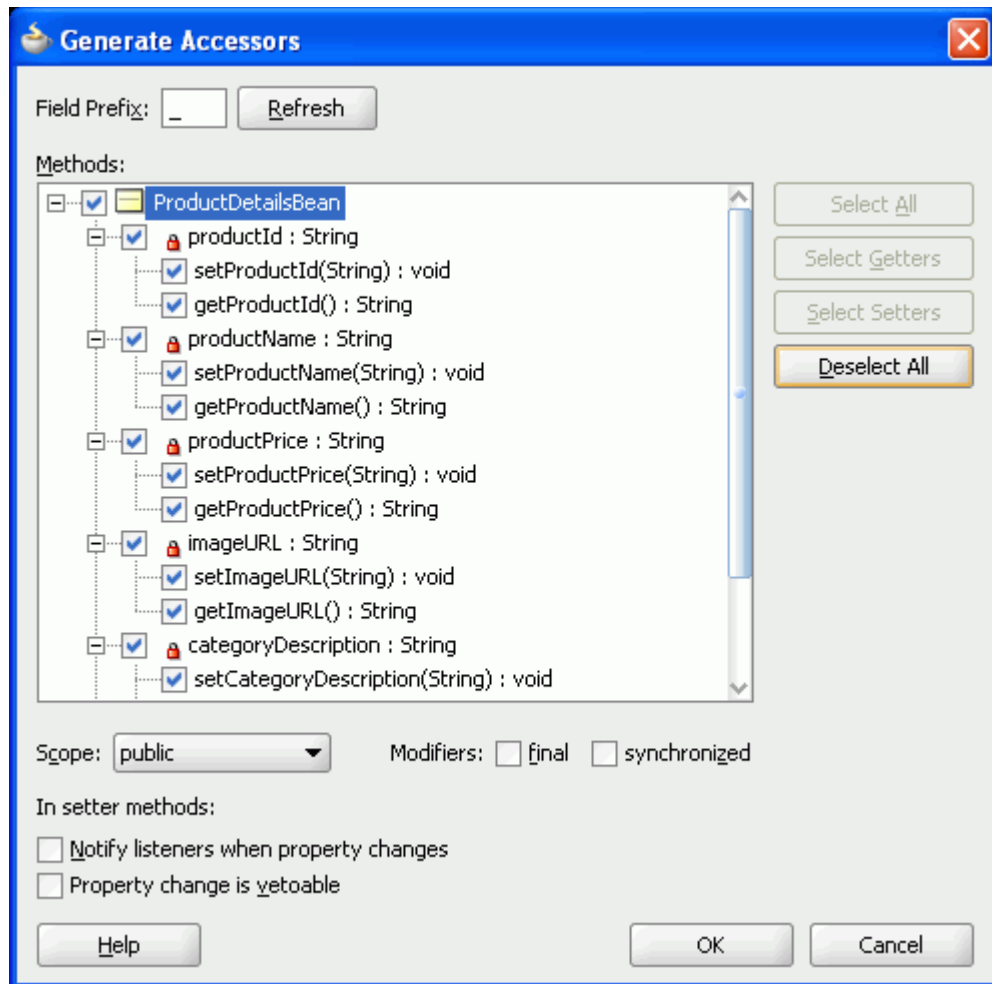
In the Structure window, right-click **ProductDetailsBean** and choose **Generate Accessors** from the context menu.

The Generate Accessors dialog box displays ([Figure 5–27](#)).

Figure 5-27 Generate Accessors Dialog Box



13. Click the **Select All** button to select all the fields you created for this bean (Figure 5-28).

Figure 5–28 Generate Accessors Dialog Box with all Fields Selected

14. Click **OK** to generate the accessors for these fields. Oracle JDeveloper generates the code as shown in [Example 5–1](#):

Example 5–1 ProductDetails JavaBean

```
package portlet;

public class ProductDetailsBean {
    private String productId;
    private String productName;
    private String productPrice;
    private String imageURL;
    private String categoryDescription;
    private String supplierName;

    public ProductDetailsBean() {
    }

    public void setProductId(String productId) {
        this.productId = productId;
    }

    public void setProductName(String productName) {
```

```
        this.productName = productName;
    }

    public String getProductName() {
        return productName;
    }

    public void setProductPrice(String productPrice) {
        this.productPrice = productPrice;
    }

    public String getProductPrice() {
        return productPrice;
    }

    public void setImageURL(String imageURL) {
        this.imageURL = imageURL;
    }

    public String getImageURL() {
        return imageURL;
    }

    public void setCategoryDescription(String categoryDescription) {
        this.categoryDescription = categoryDescription;
    }

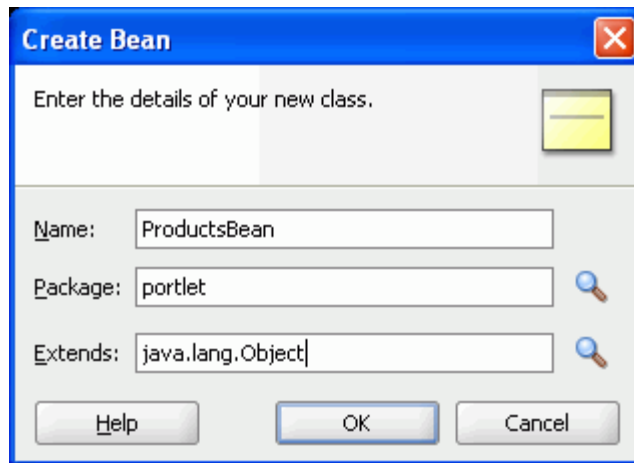
    public String getCategoryDescription() {
        return categoryDescription;
    }

    public void setSupplierName(String supplierName) {
        this.supplierName = supplierName;
    }

    public String getSupplierName() {
        return supplierName;
    }
}
```

15. Create another JavaBean for the portlet called `ProductsBean.java`. This bean represents the list of products, which the portlet will display at runtime.

To create the JavaBean right-click the **portlet** package in the Application Navigator and choose **New** from the context menu to display the Create Bean dialog box (Figure 5–29).

Figure 5–29 *Creating the ProductsBean*

16. Replace the code of `ProductsBean.java` in the Source view with the code in [Example 5–2](#).

Note: If the formatting of the code in this text does not work, you can open the `C:\TutorialContent\Portlets\ProductsBeanJava.txt` file and copy and paste the code from there.

Example 5–2 *ProductsBean.java Code*

```
package portlet;

import java.util.ArrayList;

public class ProductsBean {
    public static final String DEFAULT_PRODUCT_ID = "12";

    private ArrayList<ProductDetailsBean> products =
        new ArrayList<ProductDetailsBean>();

    public ProductsBean() {
        super();
    }

    public void addProduct(ProductDetailsBean product) {
        products.add(product);
    }

    public ArrayList<ProductDetailsBean> getProducts() {
        return products;
    }
}
```

The Source view of the bean should now look like [Figure 5–30](#).

Figure 5–30 *ProductsBean.java* Code

```

package portlet;

import java.util.ArrayList;

public class ProductsBean {
    public static final String DEFAULT_PRODUCT_ID = "12";

    private ArrayList<ProductDetailsBean> products =
        new ArrayList<ProductDetailsBean>();

    public ProductsBean() {
        super();
    }

    public void addProduct(ProductDetailsBean product) {
        products.add(product);
    }

    public ArrayList<ProductDetailsBean> getProducts() {
        return products;
    }
}

```

17. Save all your files.

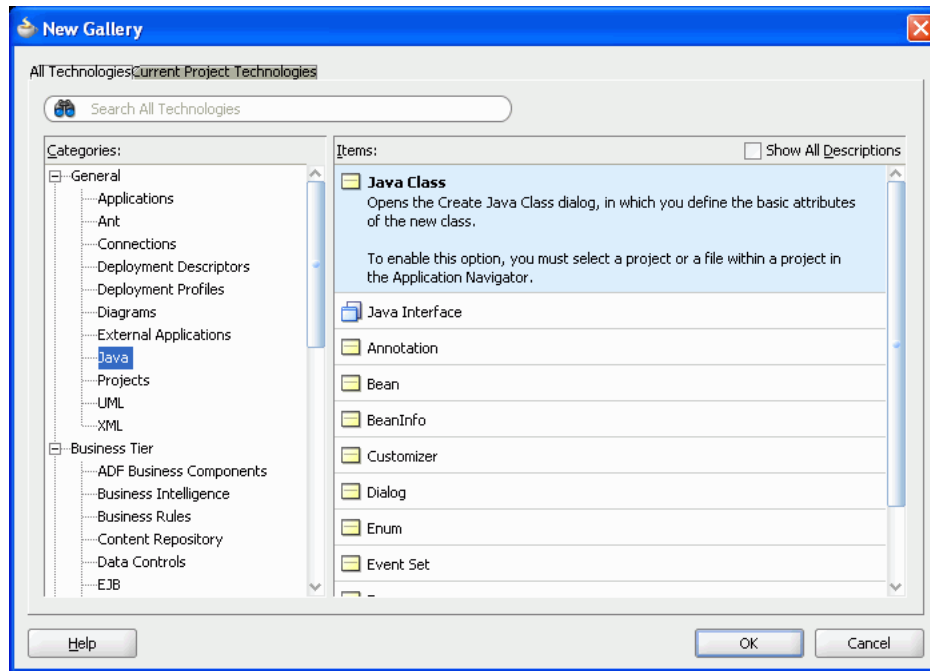
In the next steps, we'll set up the connection between our JavaBeans and the database schema that contains the data we want to show in our portlet.

Step 3: Create the Business Logic for the Standards-Based Portlet

After you create the JavaBean to access the data in the database, you create the business logic for the portlet in a Java Class. This class will contain a connection to the database, establish the connection, then query for information using the SQL statement in the class file. When you create a standards-based portlet, you must manually create this class.

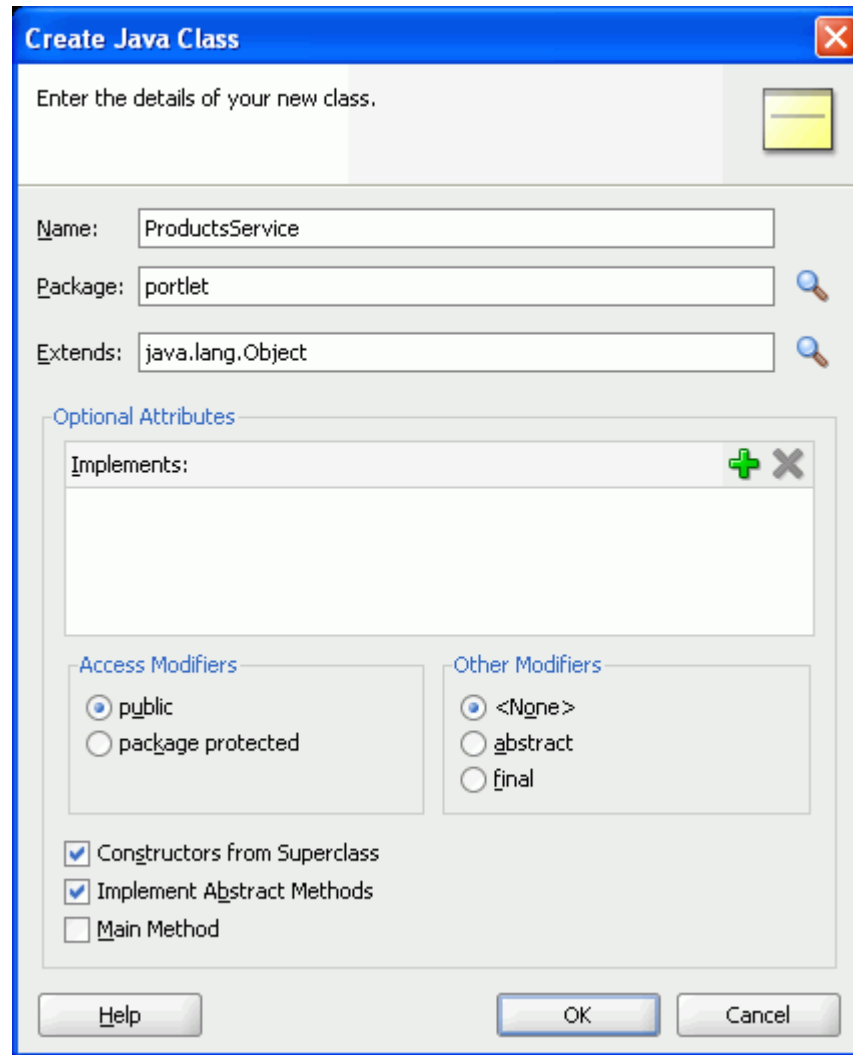
1. In the Application Navigator, right-click the **portlet** package, then choose **New** from the context menu.
2. In the New Gallery, click **Java**, then choose **Java Class** from the Items list (Figure 5–31).

Figure 5–31 Choosing the Java Class Option in the New Gallery



3. Click **OK**.
4. In the Create Java Class dialog box, in the Name field, enter the name: `ProductsService`, as shown in [Figure 5–32](#).

Figure 5–32 Create Java Class Dialog Box



5. Leave the rest of the default values in this dialog box, as we will overwrite them in the next step, then click **OK**.
6. In the `ProductService.java` file that displays, replace all the automatically-generated code with the code shown in [Example 5–3](#). The code assumes that the database where you installed the sample schema is local. In the code, modify the highlighted JDBC connection (`jdbc:oracle:thin:@localhost:1521:xe`) to point to your database and tutorial (`od`) schema

Note: If the formatting of the code in this text does not work, you can open the `C:\TutorialContent\Portlets\ProductServiceJava.txt` file and copy and paste the code from there.

Example 5–3 `ProductService.java` Code

```
package portlet;

import java.sql.Connection;
```

```

import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class ProductsService {
    public ProductsService() {
    }

    public ProductsBean getProducts() throws ClassNotFoundException {
        Connection conn = getConnection();
        ProductsBean products = new ProductsBean();
        if (conn != null) {
            try {
                Statement stmt = conn.createStatement();
                String query =
                    "SELECT DISTINCT product_id, product_name name, cost_price
price, 'http://localhost:7101/MyTutorialApplication/' || external_url image_url,
category_description, supplier_name " +
                    "FROM category_translations, products_base, suppliers" +
                    " WHERE products_base.category_id = category_
translations.category_id" +
                    " AND products_base.supplier_id = suppliers.supplier_id " +
                    " AND cost_price between 25 and 75 " +
                    " order by product_id";
                ResultSet rs = stmt.executeQuery(query);
                while (rs.next()) {
                    ProductDetailsBean productDetails = new ProductDetailsBean();
                    productDetails.setProductId(rs.getString(1));
                    productDetails.setProductName(rs.getString(2));
                    productDetails.setProductPrice(rs.getString(3));
                    productDetails.setImageURL(rs.getString(4));
                    productDetails.setCategoryDescription(rs.getString(5));
                    productDetails.setSupplierName(rs.getString(6));

                    products.addProduct(productDetails);
                }
                conn.close();
            } catch (SQLException sqle) {
                System.out.println("==== Oracle Fusion Middleware Tutorial for
WebCenter Developers ===== ");
                System.out.println("Database Connection established successfully
but encountered an error while working with the DB:" +
                    sqle);

                System.out.println("====
===== ");
            } catch (Throwable t) {
                System.out.println("==== Oracle Fusion Middleware Tutorial for
WebCenter Developers ===== ");
                System.out.println("Error while trying to get Product Details: " +
                    t);

                System.out.println("====
===== ");
            }
        }
        return products;
    }
}

```

```

    public static Connection getConnection() throws ClassNotFoundException {
        Connection conn = null;
        try {
            Class.forName("oracle.jdbc.driver.OracleDriver");
            conn =
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe", "fod",
                            "fusion");
        } catch (SQLException sqle) {
            conn = null;
            System.out.println("==== Oracle Fusion Middleware Tutorial for
WebCenter Developers ===== ");
            System.out.println("SQL error while trying to get connection to DB: "
+
                            sqle);

System.out.println("====
===== ");
        } catch (Throwable t) {
            conn = null;
            System.out.println("==== Oracle Fusion Middleware Tutorial for
WebCenter Developers ===== ");
            System.out.println("Error while trying to get the connection to DB: "
+
                            t);

System.out.println("====
===== ");
        }
        return conn;
    }
}

```

Note: In this code, the package names and import statements of the `view.jsp`, `ProductsService`, and `ProductsBean` depend on the name, class name, and package you specified or the portlet. Also, in this code, you must update the connection information to point to the database containing the tutorial (`fod`) schema. If you encounter problems with this portlet, you can check the Messages log below the Visual Editor (the Design view) to verify that the connection information you entered in this code is correct.

7. Save the file.
8. Before we can update our portlet's `view.jsp` file to use this Java class, let's return to our portlet code so that it uses the parameter value. This parameter value is the `productId` navigation parameter defined for the **Products** portlet in [Figure 5-16](#). Click the **Products.java** tab to bring it into focus. Or, if the file is not open, double-click the name in the Application Navigator.
9. In the `Products.java` file, we must update the `ProcessAction()` method in the generated portlet class file to pass the parameter value from the portlet to the Java Bean, so that the appropriate products display depending on the parameter entered.

In the `Products.java` file, locate the `processAction()` method. [Example 5-4](#) shows the section of the code in the `Products.java` file where the method is located.

Example 5-4 End of the Products.java File Containing the processAction() Method

```

public void processAction(ActionRequest request,
                        ActionResponse response) throws PortletException,
                                                    IOException {

    // Determine which action.
    String okAction = request.getParameter(OK_ACTION);
    String applyAction = request.getParameter(APPLY_ACTION);

    if (okAction != null || applyAction != null) {
        // Save the preferences.
        PortletPreferences prefs = request.getPreferences();
        String param = request.getParameter(PORTLETTITLE_KEY);
        prefs.setValues(PORTLETTITLE_KEY, buildValueArray(param));
        prefs.store();
        if (okAction != null) {
            response.setPortletMode(PortletMode.VIEW);
            response.setWindowState(WindowState.NORMAL);
        }
    }
}

```

[Example 5-4](#) shows the section we must update. For simplicity, you can replace *all* the code in the `Products.java` file with the code in the `C:\TutorialContent\Portlets\ProductsJava.txt` file. Alternatively, [Example 5-4](#) shows the updated section.

Example 5-5 Updating the Final Section of the Products.java file

```

// Form field names
public static final String PARAMETER1 = "productId";
public static final String FORM_PARAMETER1 = "form_parameter1";
public static final String FORM_SUBMIT = "dosub";
// Portlet Modes
public static final String MODE_NAME_PARAM = "mode";
public static final String MODE_VIEW = "view";

public void processAction(ActionRequest request,
                        ActionResponse response) throws PortletException,
                                                    IOException {

    // Determine what kind of action we have by examining the mode parameter
    boolean viewMode =
        MODE_VIEW.equals(request.getParameter(MODE_NAME_PARAM));

    // Extract the form field parameter and pass it through as a portlet
parameter
    String param1 = request.getParameter(FORM_PARAMETER1);
    if (param1 == null) {
        param1 = ProductsBean.DEFAULT_PRODUCT_ID;
    }

    if (viewMode) {
        // Set the new parameter values. These will be interpreted by the
        // container as navigational parameters as the names match the names
of
        // the declared parameters.
        response.setRenderParameter(PARAMETER1, param1);
    } else {
        // Determine which action.

```

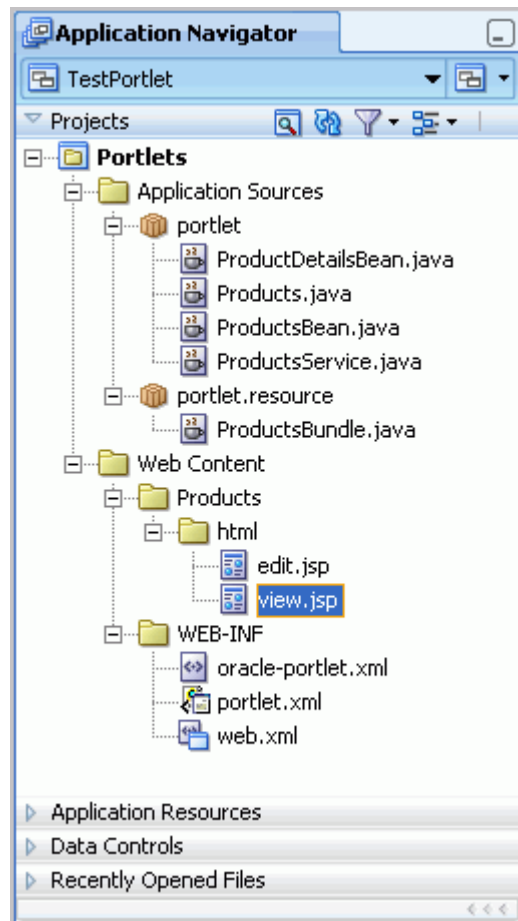
```
String okAction = request.getParameter(OK_ACTION);
String applyAction = request.getParameter(APPLY_ACTION);

if (okAction != null || applyAction != null) {
    // Save the preferences.
    PortletPreferences prefs = request.getPreferences();
    String param = request.getParameter(PORLETTITLE_KEY);
    prefs.setValues(PORLETTITLE_KEY, buildValueArray(param));
    prefs.store();
    if (okAction != null) {
        response.setPortletMode(PortletMode.VIEW);
        response.setWindowState(WindowState.NORMAL);
    }
}
}
```

10. Now that we've updated our portlet and created the Java Class to enable the portlet to communicate with the database, let's update our portlet's `view.jsp` file for the portlet to use the Java class.

In the Application Navigator, under **Portlets > Web Content > Products > html**, open the `view.jsp` file (Figure 5-33).

Figure 5-33 *View.jsp File in the Application Navigator*



11. Click the **Source** tab to view the code of this page.
12. Select all the code in the Source view and delete it.
13. Enter the code in [Example 5-6](#) in the Source view of the `view.jsp`:

Note: If the formatting of the code in this text does not work, you can open the `C:\TutorialContent\Portlets\ViewJSP.txt` file and copy and paste the code from there.

Example 5-6 View.jsp Code

```
<%@ page contentType="text/html" pageEncoding="windows-1252"
import="javax.portlet.*,
        java.util.*,
        portlet.ProductsBean,
        portlet.ProductDetailsBean,
        portlet.ProductsService,
        portlet.Products" %>
<%@ taglib uri="http://java.sun.com/portlet" prefix="portlet"%>

<!-- Include the Portlet render Response & Request objects -->
<portlet:defineObjects/>

<%
    // Get the list of products
    ProductsBean products = new ProductsService().getProducts();
    ArrayList<ProductDetailsBean> productDetails = products.getProducts();

    // "Portlet encode" the Action URL if running Portlet mode
    String actionURL = "view.jsp";
    if (renderResponse != null) {
        actionURL = renderResponse.createActionURL().toString();
    }

    // Extract the current portlet parameter value if running in Portlet mode
    String param1 = "";
    if (renderRequest != null) {
        param1 = renderRequest.getParameter(Products.PARAMETER1);
        if (param1 == null) { param1 = ""; }
    }
%>
<form method="POST" action="<%= actionURL %>">
<table>
<tr>
<th>Select</th>
<th>Product</th>
<th>Product supplied by</th>
<th>Our price</th>
</tr>

<%
for (int i = 0; i < productDetails.size(); i++) {
    ProductDetailsBean productDetail = productDetails.get(i);
%>

<tr>
<td align="center">
    <!-- Set the Form parameter name to passed as a render parameter during
processAction -->
```



```

        <input type="radio" name="<%= Products.FORM_PARAMETER1 %>"
            value="<%=productDetail.getProductId()%>"
            <%= param1.equals(productDetail.getProductId()) ? " checked='checked' "
: " " %>/>
    </td>
    <td>
        <%=productDetail.getProductName()%>
    </td>
    <td>
        <%=productDetail.getSupplierName()%>
    </td>
    <td align="right">
        $<%=productDetail.getProductPrice()%>
    </td>
</tr>

<% } %>

<tr class="PortletText1">
    <td>
        <input name="<%= Products.FORM_SUBMIT %>" type="submit"
            class="portlet-form-button" value="Show Details"></input>
    </td>
    <td colspan="3">&nbsp;&nbsp;&nbsp;</td>
</tr>

</table>

<!-- create a hidden parameter to note we're running in "view" mode -->
<input type="hidden" name="<%= Products.MODE_NAME_PARAM %>"
    value="<%= Products.MODE_VIEW %>" />
</form>

```

14. Save the view.jsp.

Now that you have established the connection to the database and set up the portlet to use the new JavaBean and Java Class to get the appropriate product information for the portlet, you are ready to include the portlet in a WAR file. WAR stands for *web archive*, and it packages all the resources, portlets, and deployment descriptors required to deploy your portlet.

Step 4: Test and Deploy the Standards-Based Portlet

In this lesson, you'll learn how to deploy the standards-based portlet to your local Default Server (the Integrated WebLogic Server). When you deploy a portlet, you package it so that it can run on a Java EE server. If you're familiar with Oracle Portal, we're in effect creating a *portlet provider*, which in the WSRP world is known as a *portlet producer*.

1. If it is not running yet, start the Default Server by choosing **Run > Start Server Instance**.
2. Before we deploy the portlet, let's quickly compile and test it. In the Application Navigator, under **Web Content > Products**, right-click **view.jsp** and choose **Run**. The portlet should compile and you should see it display in your browser, as shown in [Figure 5-34](#).

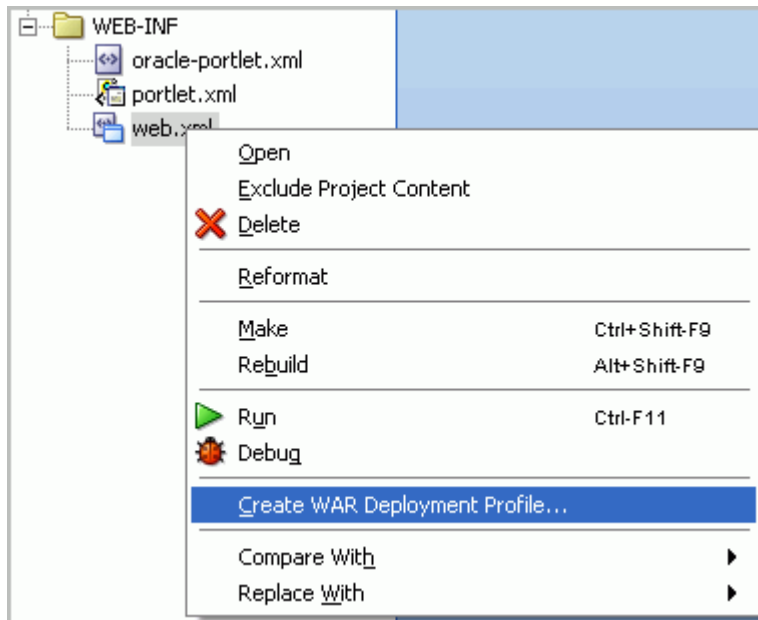
Figure 5–34 Testing the Standards-Based Portlet

Select	Product	Product supplied by	Our price
<input type="radio"/>	XBox Video Game System	Games Galore	\$75
<input type="radio"/>	Nintendo DS	Games Galore	\$55
<input type="radio"/>	Muvo Personal MP3 Player	Geeky Gadgets	\$64
<input type="radio"/>	Ipod Speakers	Transistor City	\$35
<input type="radio"/>	Ipod Shuffle 1Gb	Geeky Gadgets	\$45
<input type="radio"/>	Wii Remote	Gifts-to-Go	\$60
<input type="radio"/>	Oracle Collaboration Suite (User Perpetual) Stuffz		\$60
<input type="radio"/>	Roller Carrying Case	Z-Mart	\$50
<input type="radio"/>	Contour Carrying Case	Emporium	\$40

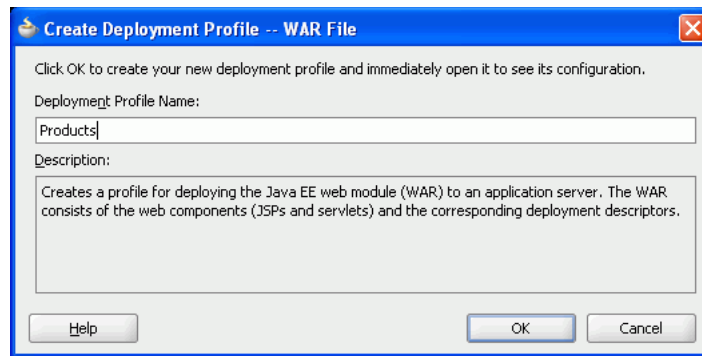
Show Details

- In the Application Navigator, under **Portlets, Web Content, WEB-INF**, right-click the `web.xml` file, then choose **Create WAR Deployment Profile** from the context menu (Figure 5–36).

Figure 5–35 Create WAR Deployment Profile Menu Option

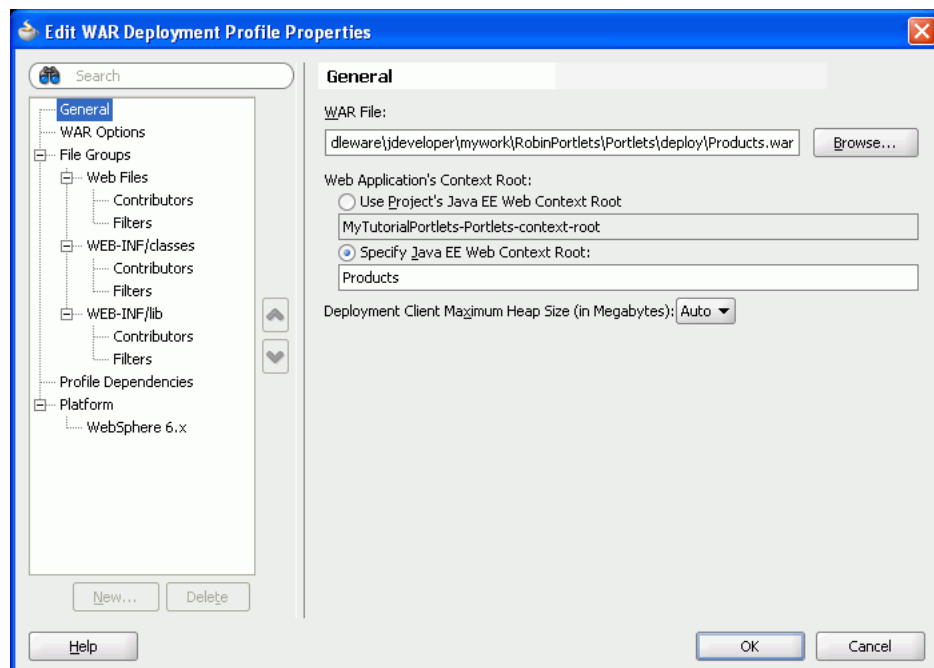


- In the Create Deployment Profile dialog box, name the Deployment Profile Products, then click **OK** (Figure 5–36).

Figure 5–36 Create Deployment Profile - WAR File Dialog Box

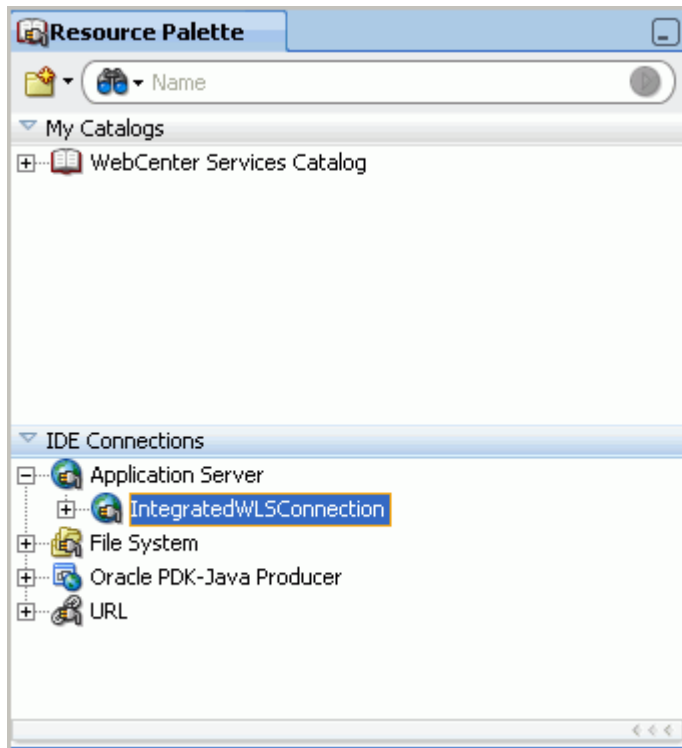
5. On the General tab of the WAR Deployment Profile Properties dialog box, look for the Web Application's Context Root setting. Let's change this to a more logical name, so that we can easily reference it later.

Select the **Specify J2EE Context Root** option, then enter `Products` in the field, as shown in [Figure 5–37](#).

Figure 5–37 WAR Deployment Profile Properties -- Setting the Context Root

6. Click **OK** to finish updating the properties, then click **OK** in the Project Properties dialog box to finish creating the WAR deployment profile.
7. Oracle WebCenter includes its own default connection to the Integrated WLS Server. You can see the connection, called **IntegratedWLSConnection**, in the Resource Palette, under IDE Connections ([Figure 5–38](#)).

Figure 5–38 IntegratedWLSConnection in the Resource Palette



8. Now, we're ready to deploy our standards-based portlet. In the Application Navigator, right-click the **Portlets** project, then choose **Deploy > Products > to > IntegratedWLSConnection** from the context menu.
9. If the Select deployment type dialog box displays, leave the default options and click **OK** (Figure 5–39).

Figure 5–39 Select Deployment Type



You can check the Deployment - Log below the Visual Editor (the Design view) for the message "Deployment finished."

- Let's go to a browser and verify whether the portlet deployment worked. In your browser, enter the URL:

`http://localhost:7101/Products/info`

The WSRP Producer Test Page displays as shown in [Figure 5–40](#).

Figure 5–40 WSRP Producer Test Page

WSRP Producer Test Page

Your WSRP Producer Contains the Following Portlets:

Portlet Name (Minimum WSRP Version)

- Products (1.0)

Container Configuration

Persistent Store Type: File
Using default value. To change it, specify the following environment entry `java:comp/env/oracle/portal/wsrp/server/persistentStore`

File Store Root: D:\Oracle\Middleware\jdeveloper\portal\portletdata
Using default value. To change it, specify the following environment entry `java:comp/env/oracle/portal/wsrp/server/fileStoreRoot`

Use Java Object Cache: false
Using default value. To change it, specify the following environment entry `java:comp/env/oracle/portal/wsrp/server/enableJavaObjectCache`

Container Version

wsrp-container.jar version is unknown

WSDL URLs

[WSRP v1 WSDL](#)
[WSRP v2 WSDL](#)

- You can choose to use either the WSRP version 1 or version 2 WSDL. In general, it's good practice to use more recent versions wherever possible. If the portlet will be consumed by WSRP 1.0 compliant consumers (such as Oracle Portal), you may want to choose WSRP 1.0.

Click **WSRP v2 WSDL** to view the XML for this WSDL ([Figure 5–41](#)).

Figure 5–41 WSDL Describing Your Portlet as a Web Service

```

- <definitions targetNamespace="urn:oasis:names:tc:wsrp:v2:wSDL">
  <import namespace="urn:oasis:names:tc:wsrp:v2:bind" location="http://localhost:7101/Products/portlets
  /wsrp2?WSDL=wsrp_v2_bindings.wsdl"/>
  <service name="WSRP_v2_Service">
    <port name="WSRP_v2_ServiceDescription_Service" binding="bind:WSRP_v2_ServiceDescription_Binding_SOAP">
      <soap:address location="http://localhost:7101/Products/portlets/WSRP_v2_ServiceDescription_Service"/>
    </port>
    <port name="WSRP_v2_PortletManagement_Service" binding="bind:WSRP_v2_PortletManagement_Binding_SOAP">
      <soap:address location="http://localhost:7101/Products/portlets/WSRP_v2_PortletManagement_Service"/>
    </port>
    <port name="WSRP_v2_Markup_Service" binding="bind:WSRP_v2_Markup_Binding_SOAP">
      <soap:address location="http://localhost:7101/Products/portlets/WSRP_v2_Markup_Service"/>
    </port>
    <port name="WSRP_v2_Registration_Service" binding="bind:WSRP_v2_Registration_Binding_SOAP">
      <soap:address location="http://localhost:7101/Products/portlets/WSRP_v2_Registration_Service"/>
    </port>
  </service>
</definitions>

```

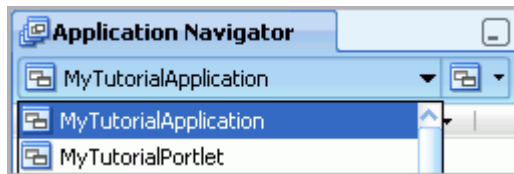
The portlet you just deployed has now been exposed as a web service. What appears in the browser is the Web Services Description Language (WSDL) that describes this web service. Now that the portlet is deployed and running, you can add this portlet to any application that can consume portlets. Our next step is to register the producer with our tutorial application, then add this portlet to MyPage.

Step 5: Register the Standards-Based Portlet with Your Application

Once we have deployed our portlet, we can register the producer with our tutorial application and add it to our page.

To register the producer with MyTutorialApplication:

1. In the Application Navigator, choose **MyTutorialApplication** from the list to return to our custom WebCenter application (Figure 5–42).

Figure 5–42 MyTutorialApplication in the Application Navigator List

2. In the Resource Palette, click the folder icon, then choose **New Connection > WSRP Producer** from the context menu.
3. On the Name page, ensure **Resource Palette** is selected, and enter a name for the producer, for example `ProductsWSRPProducer`, then click **Next** (Figure 5–43).

Figure 5–43 Register WSRP Portlet Producer -- Name

4. On the Connection page, in the WSDL URL field, enter the URL for the WSDL (typically, we use v.2):

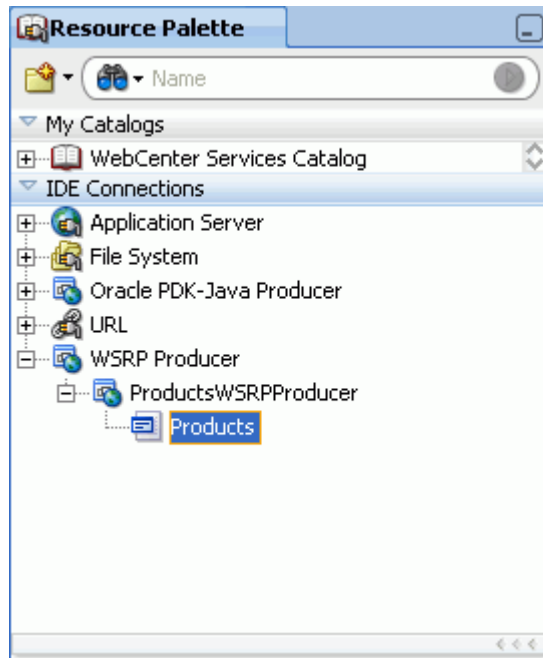
```
http://localhost:7101/Products/portlets/wsrp2?WSDL
```

Figure 5–44 Register WSRP Portlet Producer -- Connection

5. Since our Default Server is installed locally, we do not need a proxy. Click **Next** to create the connection to the WSRP Producer.

6. Let's leave the rest of the defaults and finish the wizard. On the Registration Details page, click **Finish**. You'll see a registration dialog box letting you know the registration is being completed.
7. In the Resource Palette, under IDE Connections, open the **WSRP Producer** node, then expand the **ProductsWSRPProducer** node. [Figure 5-45](#) shows the **Products** portlet here.

Figure 5-45 Products Portlet in the Resource Palette



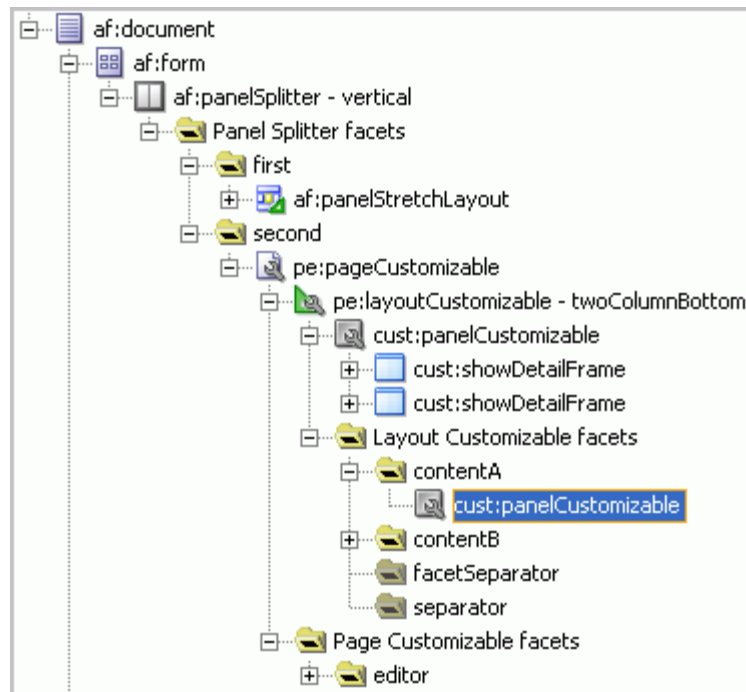
Now that we've registered the producer with our application, let's add the portlet to the page and test it.

Step 6: Test the Standards-Based Portlet in Your Application

To test the portlet, we'll add it to `MyPage.jspx`, run the page, and see if the portlet displays as expected.

1. In `MyTutorialApplication`, if `MyPage.jspx` is not open, locate the page name in the Application Navigator (under **ViewController**, **Web Content**) and double-click it.
2. In the Structure window, navigate to the second facet, then expand **Page Customizable > Layout Customizable > contentA**, and locate the **Panel Customizable**, as shown in [Figure 5-46](#).

Note: As previously mentioned, you can use the pushpin in the Structure window to freeze the current view. For this step, you should click **MyPage** in the Design view, then ensure the pushpin is in the "freeze" position (pressed).

Figure 5–46 Panel Customizable Where You Will Add the Products Portlet

3. Drag and drop the **Products** portlet from the Resource Palette onto this Panel Customizable.
4. Run the page to see how the portlet looks at runtime. This may take a few moments.

You'll notice that selecting the different options and the Show Details button do not yet work, because we have not wired this portlet with another portlet. Let's create a second portlet, then wire it to this portlet. [Figure 5–47](#) shows the Products portlet at runtime.

Figure 5–47 Products Portlet in a Browser Window

Select	Product	Product supplied by	Our price
<input type="radio"/>	XBox Video Game System	Games Galore	\$75
<input type="radio"/>	Nintendo DS	Games Galore	\$55
<input type="radio"/>	Muvo Personal MP3 Player	Geeky Gadgets	\$64
<input type="radio"/>	Ipod Speakers	Transistor City	\$35
<input type="radio"/>	Ipod Shuffle 1Gb	Geeky Gadgets	\$45
<input type="radio"/>	Wii Remote	Gifts-to-Go	\$60
<input type="radio"/>	Oracle Collaboration Suite (User Perpetual)	Stuffz	\$60
<input type="radio"/>	Roller Carrying Case	Z-Mart	\$50
<input type="radio"/>	Contour Carrying Case	Emporium	\$40

Show Details

Step 7: Register the Preconfigured Portlet Producer

For this tutorial, we will use a preconfigured portlet producer, called OmniPortlet, which is predeployed to the Integrated WLS (WebLogic) Server, also referred to as the Default Server. Be sure you have followed the steps in [Chapter 2, "Preparing for the Tutorial"](#) before you proceed.

To register our portlet producer:

1. Ensure the Default Server is running.
2. To find out the URL for the OmniPortlet producer running on the Default Server, choose **Help > WebCenter Preconfigured Server Readme**. This Read Me file shows all the information needed to use the portlets that are predeployed to the Default Server, and the login credentials for the server.
3. In the Read Me file, under **Preconfigured Portlet Producers**, then **PortalTools Portlet Producers**, click **OmniPortlet Producer**.
4. In your browser, you should see the OmniPortlet Producer Test Page. Copy the URL from the location bar. The URL should look like this:

```
http://localhost:7101/portalTools/omniPortlet/providers/omniPortlet
```

Entering this URL in your browser displays the OmniPortlet producer test page ([Figure 5-48](#)).

Figure 5-48 OmniPortlet Producer Test Page

The screenshot displays the Oracle Application Server Portal Producer Test Page for the omniPortlet. The page is titled "Producer Test Page: omniPortlet" and includes several sections:

- Portlet Information:** Lists the portlets contained in the producer: OmniPortlet and Simple Parameter Form.
- Producer Initialization Parameters:** Shows parameters defined in the web application configuration file (web.xml):

Name	Value
invalidation_caching	true
- Producer Configuration:** Provides settings and their status:

Setting	Status
HTTP Proxy <small>To change settings, edit <proxyInfo> tag in both OmniPortlet's and Web Clipping's provider.xml files.</small>	Not configured
BI Graph Bean	Installed
- Repository Setting:** Shows the status of the Secured Data Repository:

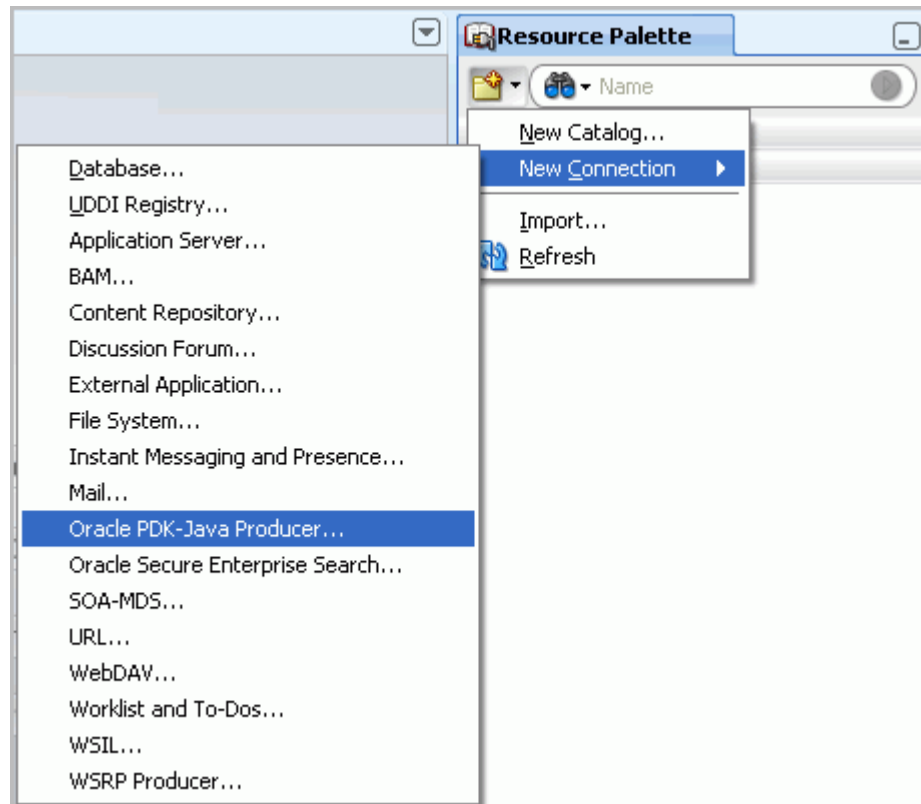
Repository Setting	Status
Secured Data Repository <small>Security will not work if the repository is not configured. To change settings, edit <repositoryInfo> tag in Web Clipping's provider.xml.</small>	Configured
- Available Extensions:** Lists registered extensions:

Name	Type
webpage	Data Source
HTML	Layout
Parameter Form	Layout

The page also features a "Home" link at the bottom center.

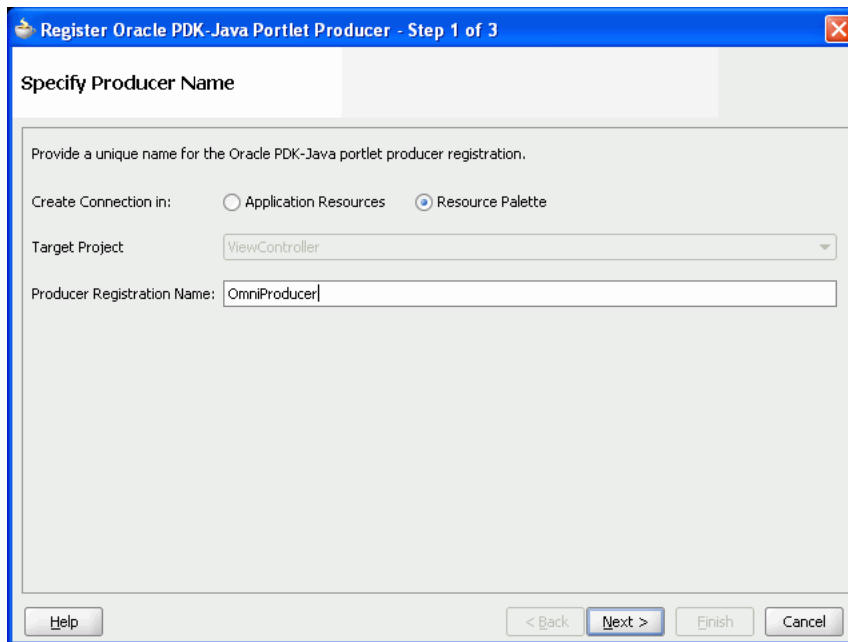
5. In the Resource Palette, click the **New** icon next to the search toolbar, then choose **New Connection**, and then **Oracle PDK-Java Producer** (Figure 5-49).

Figure 5-49 Registering an Oracle PDK-Java Producer



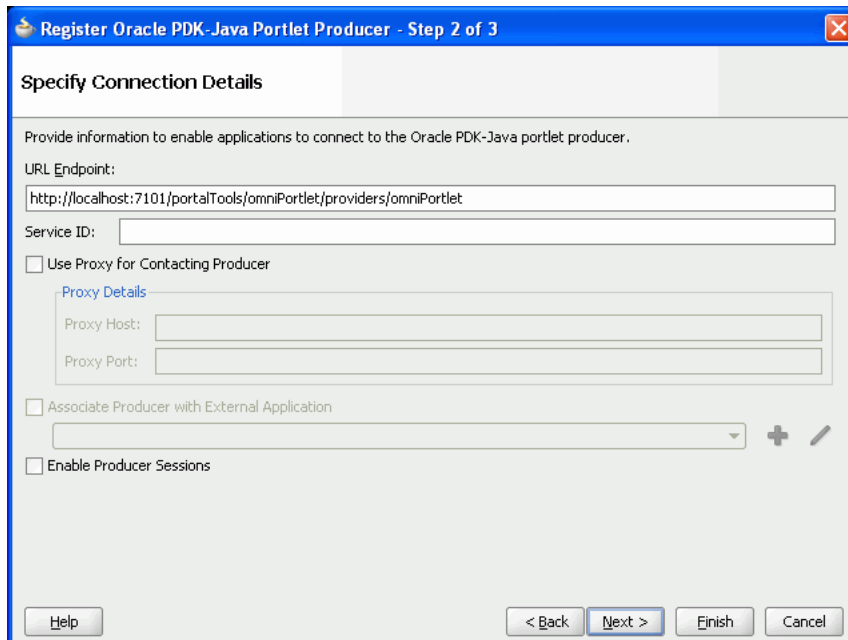
6. On Step 1 of the Register Oracle PDK-Java Portlet Producer wizard, let's enable this portlet producer to be used across all our applications. Select **Resource Palette**.
7. In the Name field, enter `OmniProducer` (Figure 5-50).

Figure 5–50 Naming the OmniPortlet Producer



8. Click **Next**.
9. Enter this URL for the OmniPortlet Producer Test Page, which you copied earlier in the **URL Endpoint** field (Figure 5–51).

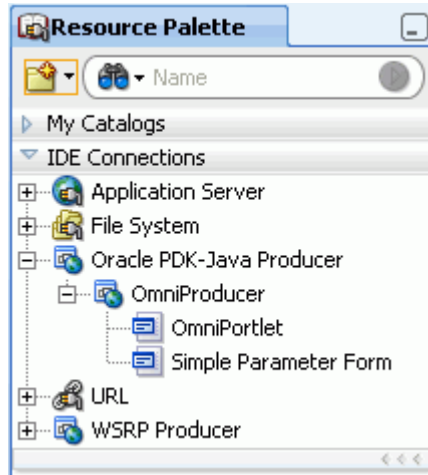
Figure 5–51 Specifying the Connection Details for the OmniPortlet Producer



10. Click **Next**.
11. On the Registration Details page, click **Finish**. You should see a message indicating that Oracle JDeveloper is registering your producer.

Because you chose to register the portlet producer in the Resource Palette, your new portlet producer displays in the IDE Connections list of the Resource Palette, as shown in [Figure 5-52](#).

Figure 5-52 *OmniProducer in the IDE Connections List*



Step 8: Add an OmniPortlet to Your Page

One type of portlet you can use with Oracle WebCenter Framework is OmniPortlet. This portlet is provided out of the box, and is preconfigured on the Default Server. It lets you quickly create portlets using a variety of default layouts and data sources.

[Figure 5-53](#) shows a sample portlet you can build with OmniPortlet. The portlet displays items stored in a database as images, which are stored locally. After you wire this portlet with the portlet you created in "[Step 1: Create a Standards-Based Java \(JSR 168\) Portlet](#)", a user can click these images in your application to learn more about that item.

Figure 5-53 *OmniPortlet at Runtime*



Before you can use this OmniPortlet, ensure that you have added the sample schema to your database and that you have started the Default Server as described in [Chapter 2, "Preparing for the Tutorial."](#) Also, ensure that you have registered the OmniPortlet producer, as described in [Step 7: Register the Preconfigured Portlet Producer](#).

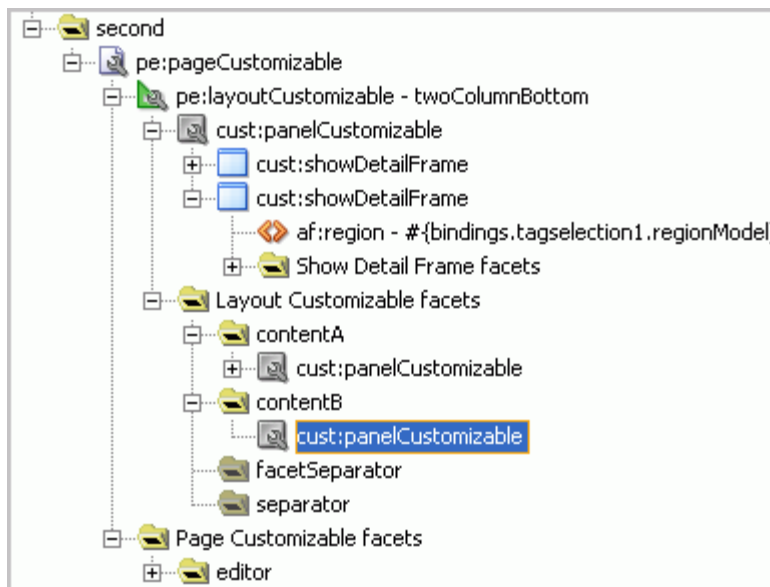
1. In Oracle JDeveloper, ensure **MyTutorialApplication** is open.

2. In the Application Navigator, ensure the `MyPage .jspx` is in the Design view. If you've closed your application, you can double-click the page name in the Application Navigator.
3. In the Structure window, locate the Panel Customizable in second facet, where you added the Documents and Tags services.

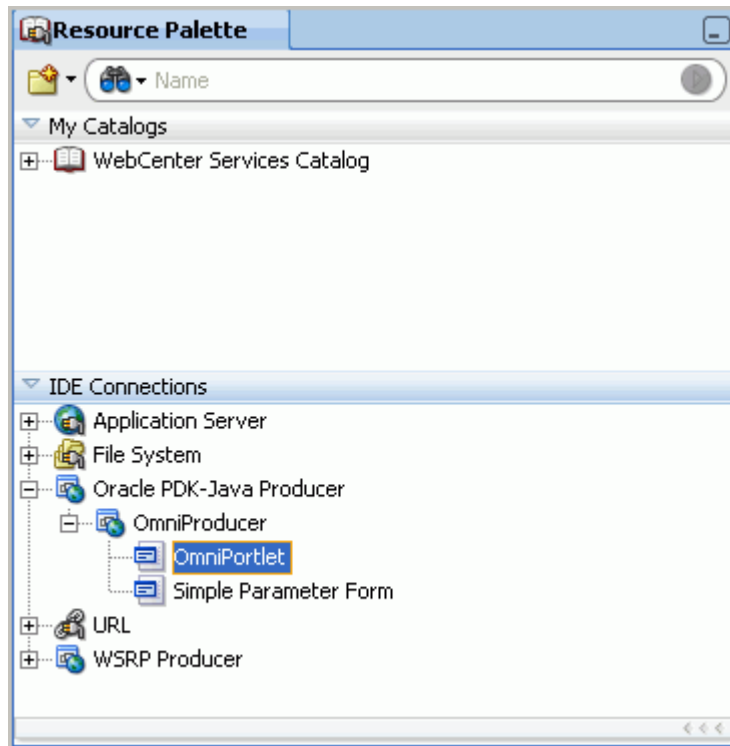
Note: As previously mentioned, you can use the pushpin in the Structure window to freeze the current view. For this step, you should click **MyPage** in the Design view, then ensure the pushpin is in the "freeze" position (pressed).

4. Open the Layout Customizable facets, then open `contentB` to expose the Panel Customizable (you added the standards-based Java (JSR 168) portlet to `contentA`), as shown in [Figure 5-54](#).

Figure 5-54 Panel Customizable in contentB

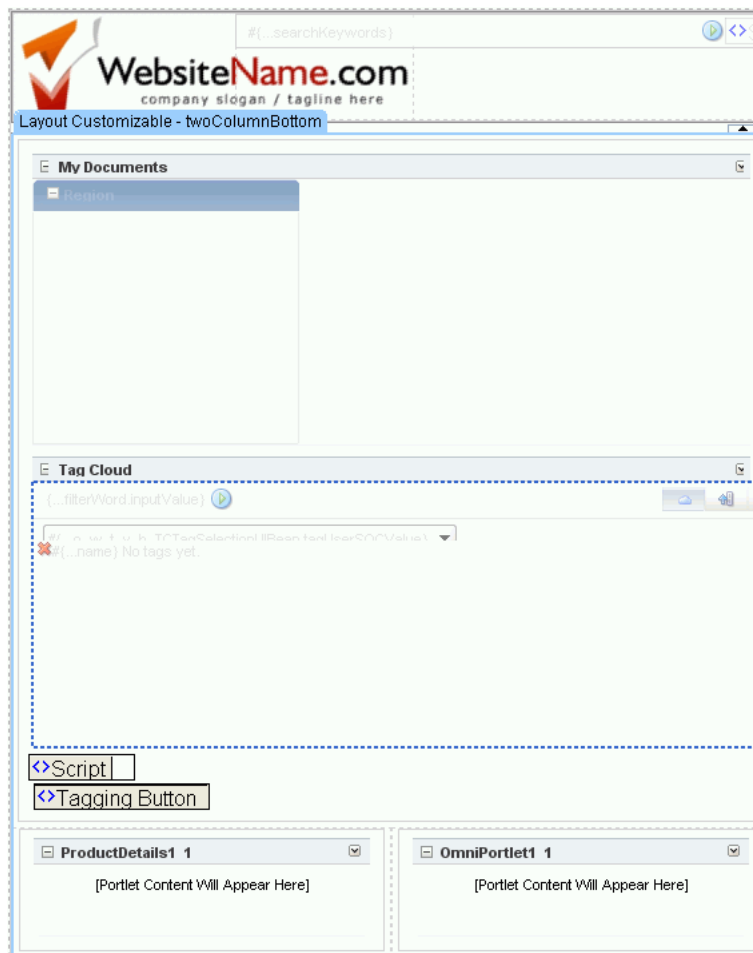


5. In the Resource Palette, under IDE Connections, navigate to **Oracle PDK-Java Producer > OmniProducer > OmniPortlet** ([Figure 5-55](#)).

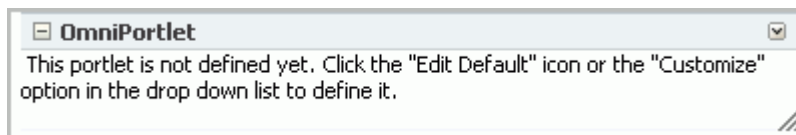
Figure 5–55 *OmniPortlet in the Resource Palette*

6. Drag and drop **OmniPortlet** onto your page onto the **Panel Customizable** under contentB in the Structure window. OmniPortlet displays in the Design view of your page, as shown in [Figure 5–56](#). If you see an error message, ensure you have started the Default Server, then try again.

For more troubleshooting tips on using OmniPortlet, refer to Oracle Fusion Middleware Developer's Guide for Oracle WebCenter.

Figure 5–56 OmniPortlet Instance on MyPage at Design Time

- Now that we've added OmniPortlet to our page, let's customize its contents. Run your page to the browser. [Figure 5–57](#) shows the OmniPortlet on MyPage in your browser.

Figure 5–57 Undefined OmniPortlet at Runtime

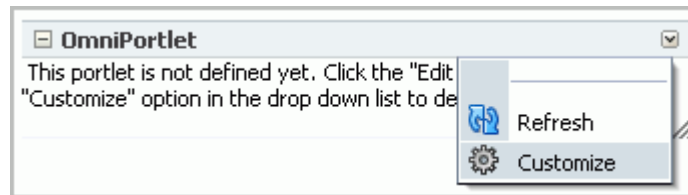
Step 9: Define OmniPortlet at Runtime

In this section, we'll customize our OmniPortlet to bring in some information from the database schema and the images we added to our application in [Chapter 2, "Preparing for the Tutorial."](#)

Since OmniPortlet is a preconfigured portlet, our only steps are to add it to our application, then customize it at runtime. Now that we've placed it on our page and run our page to the browser, let's customize the layout and content of this portlet.

1. In the upper right corner of the portlet, click the arrow, and choose **Customize** to launch the OmniPortlet wizard (Figure 5-59).

Figure 5-58 Customize Link for OmniPortlet



2. On the Data Type page, choose **SQL** so that we can obtain data from the schema in a database, then click **Next**. At this point, be sure you've added the schema to your database (as described in [Chapter 2, "Preparing for the Tutorial"](#)) and the images to your application resources (as described in [Chapter 3, "Creating a WebCenter Application with a Customizable Page"](#)) otherwise the portlet will not retrieve the sample tutorial data.

If you're familiar with SQL and your database, you can always use your own sample data, but the ensuing images and steps may not necessarily be accurate for you.

3. On the Data Source page, we can define our SQL statement and set up the connection to the database where we installed the schema.

In the Statement box, enter the code in [Example 5-7](#). The parameter in the statement will be used when we wire this portlet with the Products portlet we added in [Step 5: Register the Standards-Based Portlet with Your Application](#).

Note: If the formatting of the code in this text does not work, you can open the `C:\TutorialContent\Portlets\OmniPortlet_SQL_Statement.txt` file and copy and paste the code from there.

Example 5-7 OmniPortlet SQL Statement

```
SELECT product_name name
, cost_price our_price
, cost_price * 1.3 retail_price
, cost_price * 0.3 savings
, external_url image_url
, category_description
, supplier_name
FROM category_translations, products_base, suppliers
WHERE products_base.category_id = category_translations.category_id
AND products_base.supplier_id = suppliers.supplier_id
AND products_base.product_id = nvl('##Param1##',0)
```

Note: In the statement, you'll notice a reference to the localhost. This refers to the OmniPortlet producer you registered in [Chapter 2, "Preparing for the Tutorial."](#) Notice that you can also click the **Test** button to see what the statement will return.

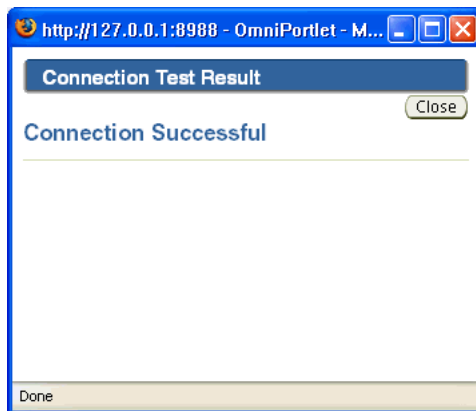
4. Under Connection, click **Edit Connection**.

- Enter a name for your connection and the connection information for your database. The schema username is `fod` and the password is `fusion`. For example, if you are using an Oracle XE database locally, the page would look like [Figure 5–59](#).

Figure 5–59 Connection Information for OmniPortlet

- Let's make sure the connection information is correct by clicking **Test**. [Figure 5–60](#) shows a successful test message.

Figure 5–60 Successful Connection Message





- Click **Close**, then click **OK** to finish creating the connection.
- Under Portlet Parameters, set the Default Value for **Param1** to 12. Doing so provides a default value for the parameter in our SQL statement.

[Figure 5–61](#) shows the SQL Source page.


Figure 5–61 SQL Source Page

SQL

Statement

```
SELECT product_name name
, cost_price our_price
, cost_price * 1.3 retail_price
, cost_price * 0.3 savings
, external_url image_url
, category_description
, supplier_name
```

 **TIP** You can use ##ParamN## (ex: ##Param1##) in place of any text in the SQL Query. You can also use :variable (ex: deptno = :p_dept) to define bind variables. [Learn more...](#)

Connection

To access secured data, you will need to provide connection information to the data source.

Connection Information fod (FOD)

Use this connection information for all users
 User must re-enter connection information

Portlet Parameters

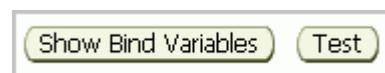
Parameters are passed to the portlet from the page when the portlet is displayed. These parameters can be mapped to page level parameters by editing the Page Properties.

Parameter Name	Default Value	Personalizable	Personalize Page Label	Personalize Page Description
Param1	12	<input type="checkbox"/>	Param1	Description for Parameter
Param2		<input type="checkbox"/>	Param2	Description for Parameter
Param3		<input type="checkbox"/>	Param3	Description for Parameter
Param4		<input type="checkbox"/>	Param4	Description for Parameter
Param5		<input type="checkbox"/>	Param5	Description for Parameter

Note: If you do not have the `images` folder in your ViewController project as described in [Chapter 3, "Creating a WebCenter Application with a Customizable Page,"](#) the images referenced in the SQL statement will not display. You can go back and add these now by following [Step 2: Add the Images Files to the Application](#), but you must run the application to your browser again in order for OmniPortlet to recognize the new folder.

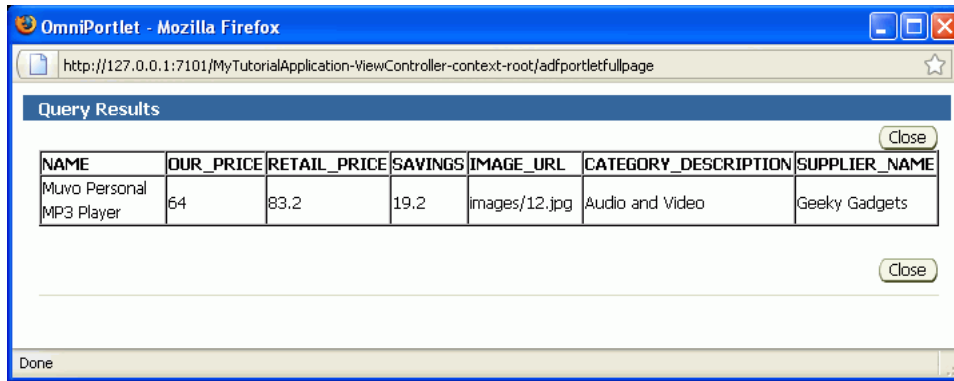
9. Click **Test** next to the Show Bind Variables button to validate the SQL statement and connection.

Figure 5–62 Test Button



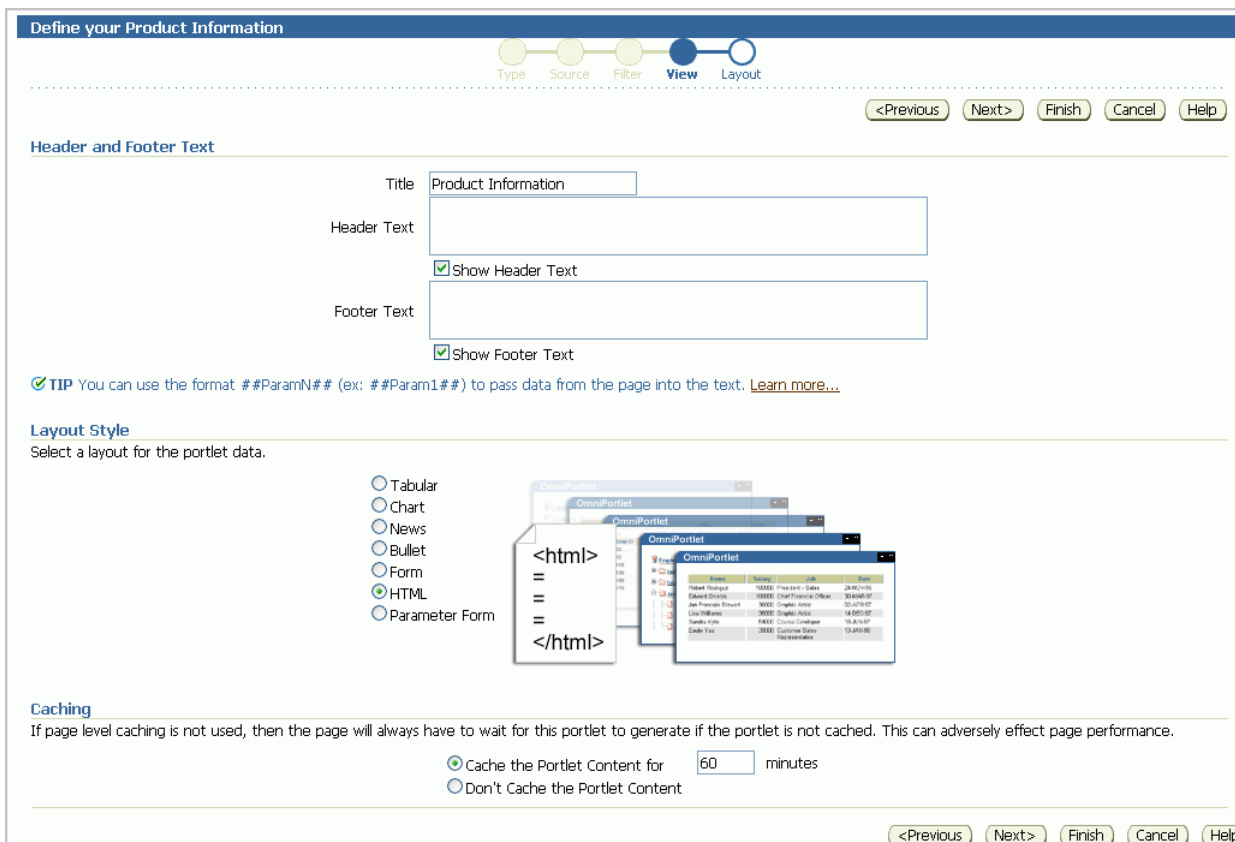
A pop-up window displays returning a row based on the statement. If you do not see a row returned, validate your SQL statement, connection, and portlet parameter.

Figure 5–63 SQL Statement Test Results



10. Click Next.
11. On the Filter page, click Next.
12. On the View page, let's name the portlet, for example `Product Information`, by entering the name in the **Title** field.
13. We can choose any of the default layouts for this portlet. However, let's check out the HTML layout, which you can use to fine tune the look and feel of your portlet. Under **Layout Style**, select **HTML**, as shown in [Figure 5–64](#), then click Next.

Figure 5–64 View Page



14. On the Layout page, you'll notice a form for filling in HTML for the template. Here, you can modify the layout of your portlet by updating the Header, Body, and Footer fields. You can use the default layout that OmniPortlet provides, but let's step through creating our own HTML layout.

From the Quick Start list, select **Clear Fields**, then click **Apply**. Doing so removes the existing HTML code from the layout template.

15. In the Repeating Section, enter the HTML in [Example 5-8](#) to create a table that formats the data.

Note: If the formatting of the code in this text does not work, you can open the `C:\TutorialContent\Portlets\OmniPortlet_HTML_Layout.txt` file and copy and paste the code from there.

Example 5-8 *OmniPortlet HTML Layout Code*

```
<table>
<tr>
<td>
<font class="PortletHeading1">##NAME##</font>
</img>
</td>
<td>
<table>
<tr class="PortletText2">
<td>
Product supplied by
<font class="PortletHeading3">
##SUPPLIER_NAME##
</font>
</td>
</tr>
<tr class="PortletText2">
<td>
List price: $##RETAIL_PRICE##
</td>
</tr>
<tr class="PortletHeading2">
<td>
Our price: $##OUR_PRICE##
</td>
</tr>
<tr class="PortletText2">
<td>
You save: $##SAVINGS##
</td>
</tr>
<tr class="PortletText2">
<td>
<font class="PortletHeading2">Availability:</font>
In Stock. Ships from and sold by
<font class="PortletHeading2">FusionOnline.com</font>
. Gift-wrap available.
</td>
</tr>
<tr>
<td>&nbsp;</td>
```

```

        </tr>
    </table>
</td>
</tr>
<tr class="PortletText1">
    <td colspan="2">Want it delivered Thursday? Order it in the next 22 hours and
        45 minutes, and choose One-Day Shipping at checkout.</td>
</tr>
</table>

```

The Layout page should now look like [Figure 5–65](#).

Figure 5–65 HTML Layout Page

Quick Start
Use the Quick Start to populate the sections below with starting HTML code.

Quick Start Clear Fields

Tip Clicking the Apply button permanently deletes any content currently in the below sections and replaces it with the template code.

Non-Repeating Heading Section
Insert data label or system variable into text area:

Tip HTML code entered in the above section displays after the contents of the Header text box on the View tab.

Repeating Section
Insert data column or system variable into text area:

```

<table>
<tr>
<td>
<font class="PortletHeading1">##NAME##</font>
</img>

```

Tip HTML code entered in the above section displays once for each data record.

Non-Repeating Footer Section
Insert data label or system variable into text area:

Tip HTML code entered in the above section displays before the contents of the Footer text box on the View tab.

16. Click **Finish**. [Figure 5–66](#) shows the OmniPortlet in your browser.

Figure 5–66 Completed OmniPortlet at Runtime



Now that we have created our two portlets, let's wire them.

Step 10: Wire the Standards-Based Portlet and OmniPortlet Together

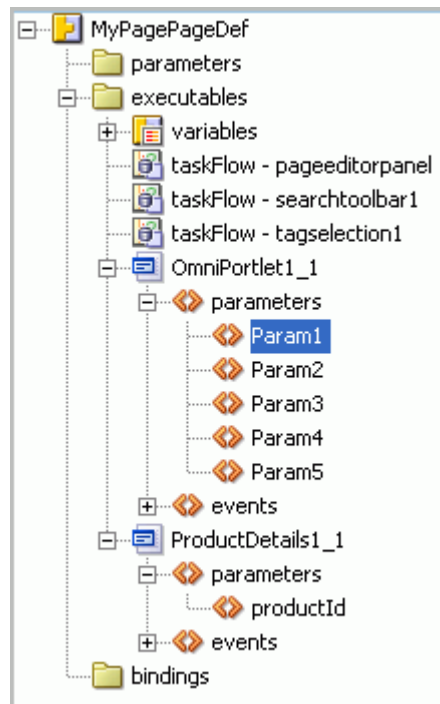
When you added the code for the standards-based portlet called `Products`, you also included a parameter called `productId`. When you select different options in the `Products` portlet, the application will send this parameter to `OmniPortlet` so that it can display the details of a particular product with that product identification number.

We now must map the two parameters to each other, so that when you choose an option in the `Products` portlet, the information in `OmniPortlet` updates accordingly.

To wire the portlets:

1. In Oracle JDeveloper, ensure **MyTutorialApplication** is open.
2. In the Application Navigator, open the page definition file for the `MyPage.jspx` page. You can do this by opening the page definition file itself in the Application Navigator, or by right-clicking the page and choosing **Go to Page Definition**.
3. In the Structure window, use the pushpin to freeze the current view. Ensure the `MyPage` Page Definition is selected in the Design view, then, in the Structure window, click the pushpin so that it is in the "freeze" position (pressed).
4. In the Structure window for the `MyPagePageDef.xml` (page definition) file, expand **executables**, expand **OmniPortlet1_1** > **parameters**, then select the portlet variable **Param1**, as shown in [Figure 5-67](#).

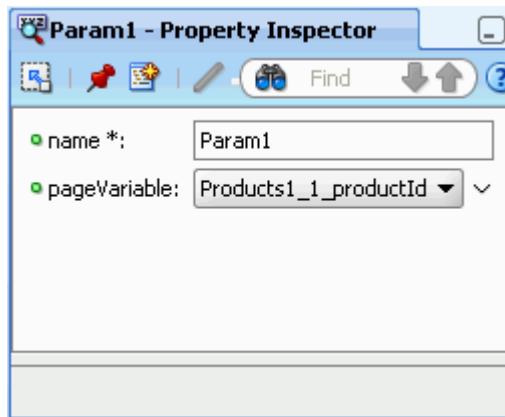
Figure 5-67 *OmniPortlet Variable in the Structure Window*



5. While the variable is selected, you should be able to view the properties for it in the Property Inspector.

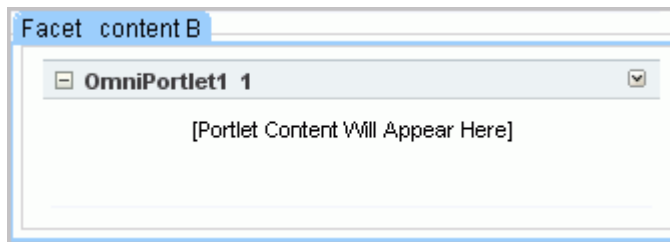
Set the **pageVariable** property to **Products1_1_productId**, as shown in [Figure 5-68](#).

Figure 5–68 *Setting the Default Value*



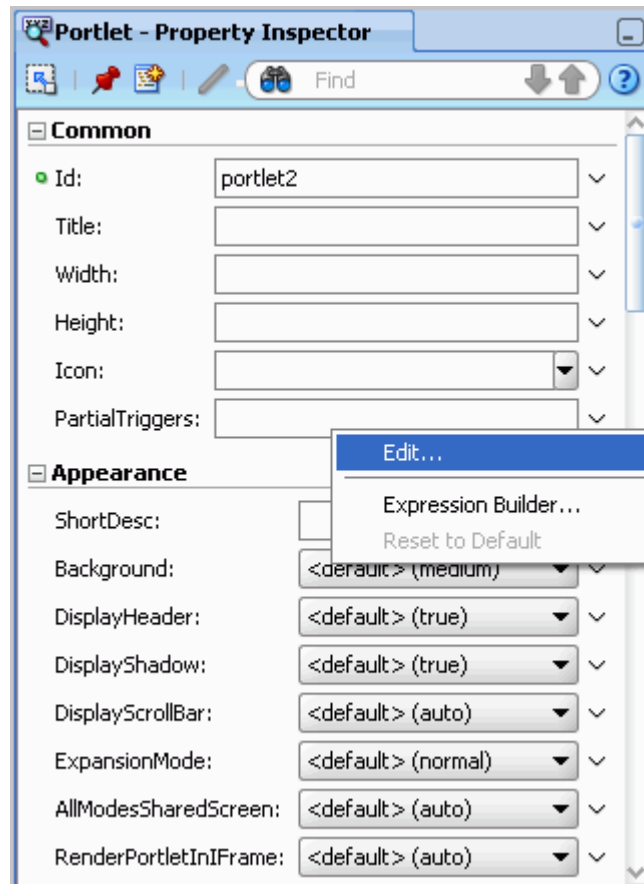
6. Now we want the detail portlet (OmniPortlet) to refresh whenever the value from the master portlet changes. To do so, we add a Partial Trigger to the detail portlet. Click the **MyPage.jspx** tab at the top of the Visual Editor to bring it into focus.
7. Select the **OmniPortlet** on the page in the Design view.

Figure 5–69 *OmniPortlet on MyPage in the Design View*



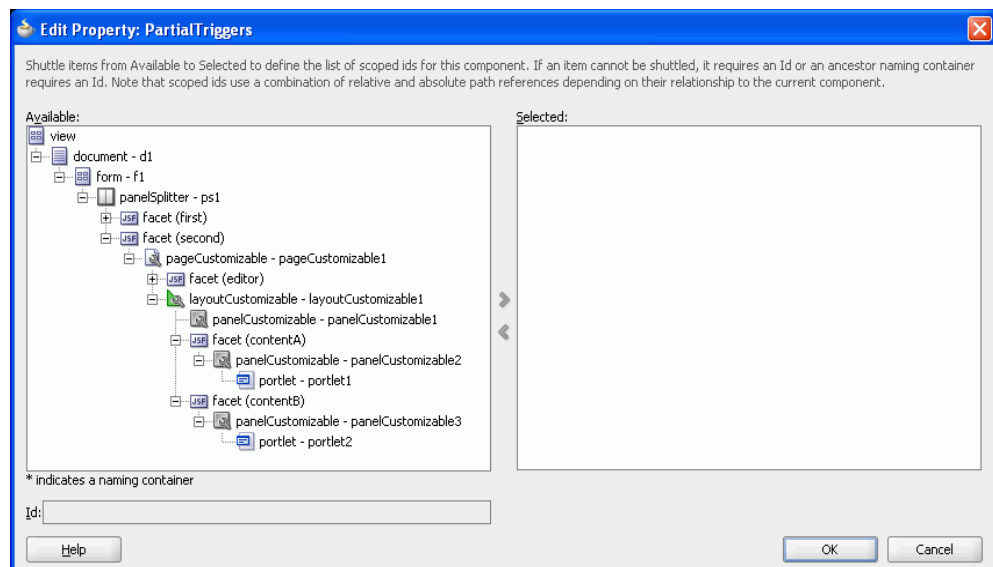
8. In the Property Inspector for OmniPortlet, under Common, click the arrow next to **PartialTriggers**, then choose **Edit** (Figure 5–70).

Figure 5–70 Editing the PartialTriggers Property for OmniPortlet



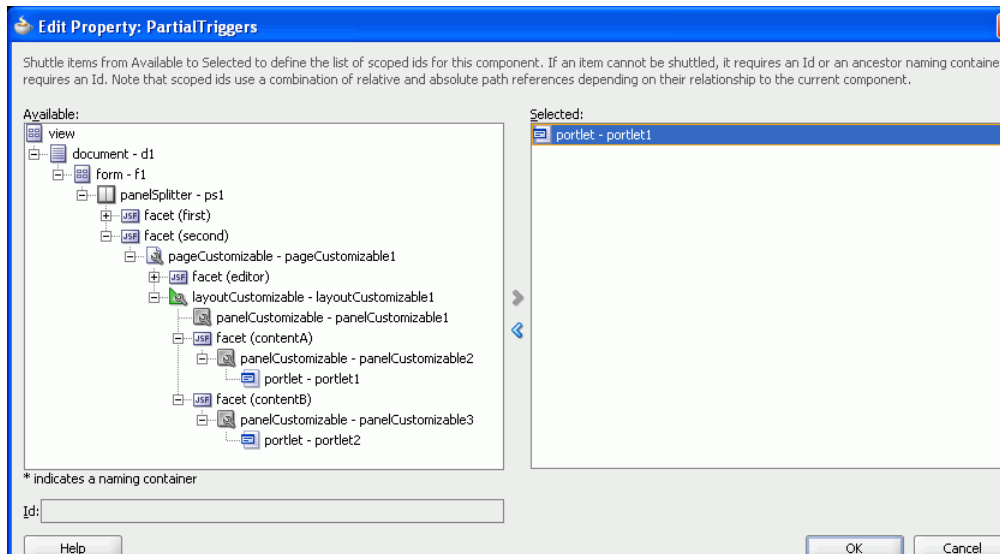
- In the Edit Property: PartialTriggers dialog box, the Portlet ID for OmniPortlet is automatically selected. Locate the **Portlet ID** for the Products portlet, which is in facet (contentA), under the Panel Customizable (Figure 5–71).

Figure 5–71 Locating the Portlet ID for the Products Portlet



10. Select the Portlet ID, in this case **portlet - portlet1**, and click the right arrow to move it to the Selected list, then click **OK** (Figure 5-72).

Figure 5-72 Selecting the Portlet ID for the Products Portlet



11. Now that we've wired the portlet parameters, let's examine how they behave at runtime.

Run **MyPage.jspx** to your browser. In the next step, we'll test how the JSR 168 (Products) portlet and the OmniPortlet interact at runtime.

Step 11: Test the Interaction Between the Portlets

Let's test the portlets at runtime.

1. In the Products portlet, select an option, for example **iPod Speakers**, then click **Show Details**.
2. In the OmniPortlet, notice that the portlet updates to display information about the iPod speakers (Figure 5-73).

Figure 5-73 Testing the Interaction Between the Portlets

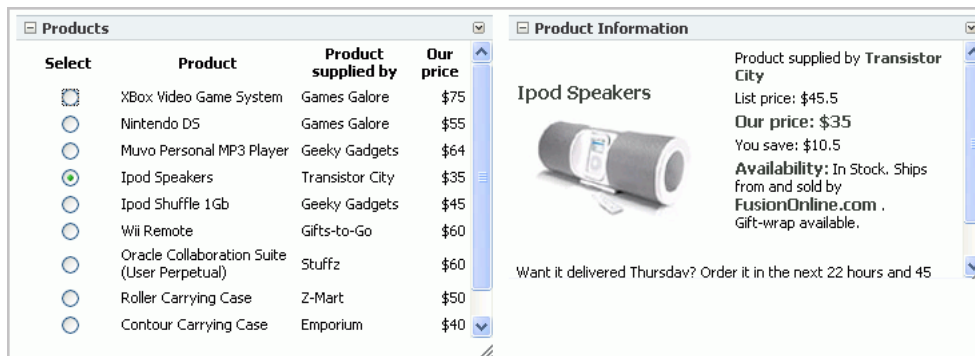


Figure 5-74 shows MyPage in your browser.

Figure 5-74 MyPage at Runtime

The screenshot displays a web application interface for 'WebsiteName.com'. The main content area is divided into several portlets:

- My Documents:** A file manager showing a 'Manuals' folder containing sub-folders 'prodA', 'prodB', and 'prodC'. The current location is 'Manuals'.
- Tag Cloud:** A cloud of tags including '650 adaptor bluetooth creative digital document entertainment flashphone fod guide home important iphone ipodvideo lkj media microsoft myheadset myphone myplayer myspeaker myspeakers nano palm phone player prodb producta productb television'.
- Products:** A table listing various products with their suppliers and prices.
- Product Information:** A detailed view for the 'Muvo Personal MP3 Player', showing its price, availability, and shipping options.

Select	Product	Product supplied by	Our price
<input type="radio"/>	XBox Video Game System	Games Galore	\$75
<input type="radio"/>	Nintendo DS	Games Galore	\$55
<input type="radio"/>	Muvo Personal MP3 Player	Geeky Gadgets	\$64
<input type="radio"/>	Ipod Speakers	Transistor City	\$35
<input type="radio"/>	Ipod Shuffle 1Gb	Geeky Gadgets	\$45
<input type="radio"/>	Wii Remote	Gifts-to-Go	\$60
<input type="radio"/>	Oracle Collaboration Suite (User Perpetual)	Stuffz	\$60
<input type="radio"/>	Roller Carrying Case	Z-Mart	\$50
<input type="radio"/>	Contour Carrying Case	Emporium	\$40

Product Information for Muvo Personal MP3 Player:

- Product supplied by: Geeky Gadgets
- List price: \$83.2
- Our price: \$64**
- You save: \$19.2
- Availability:** In Stock. Ships from and sold by FusionOnline.com . Gift-wrap available.

Want it delivered Thursday? Order it in the next 22 hours and 45 minutes, and choose One-Day Shipping at checkout.

Congratulations! You've completed this lesson and created two portlets and made them communicate with each other. Continue on to [Chapter 6, "Conclusion"](#) to review what you learned in this tutorial, and where you can find more information about the features you used.

Conclusion

Congratulations! You have created a custom WebCenter application and learned about the fundamentals of Oracle WebCenter Framework.

Summary

In this Tutorial, you learned how to perform a few quick and easy steps to create a custom WebCenter application. You also learned about a few components of Oracle WebCenter Framework, including Oracle Composer and the WebCenter Web 2.0 Services.

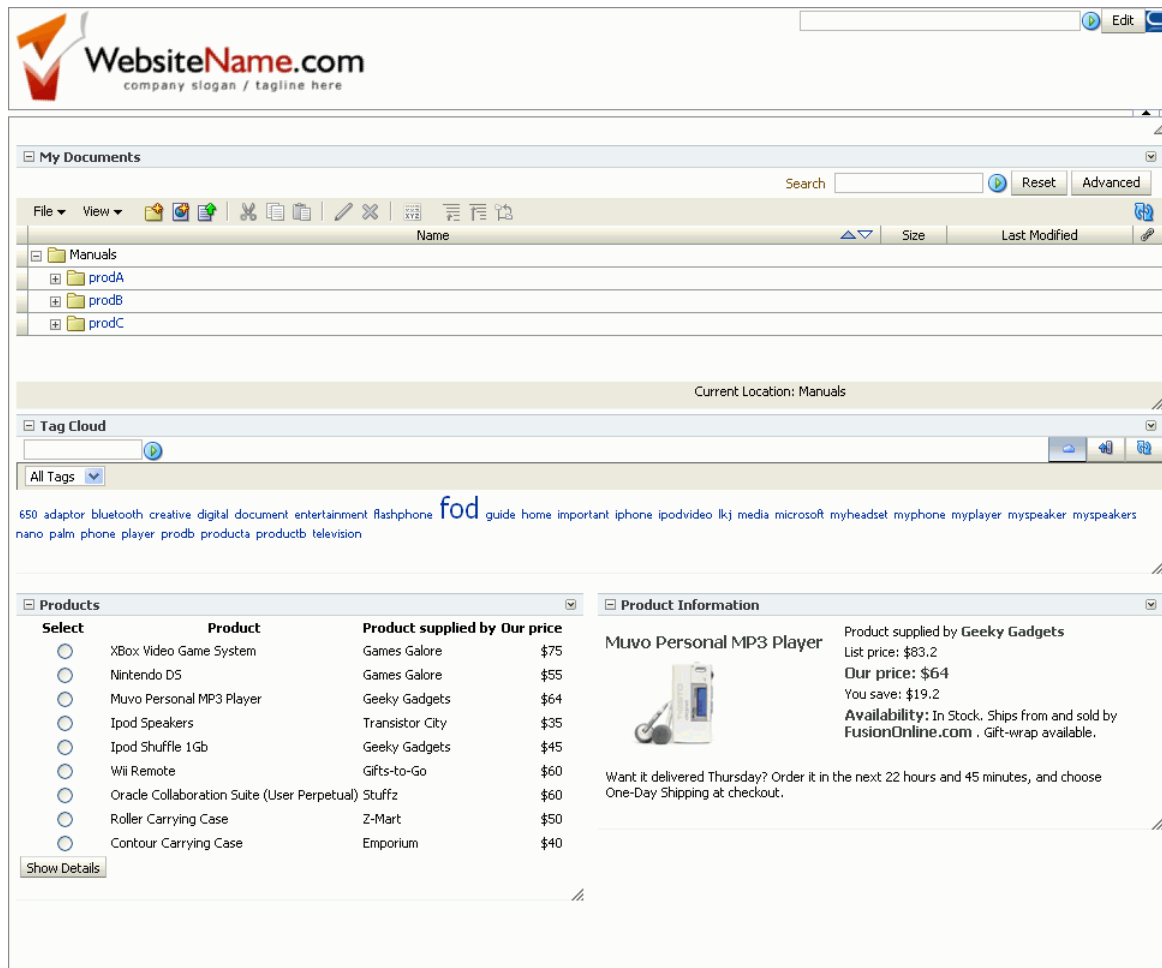
Specifically, you learned how to:

- Create a database connection, which allowed you to access a database containing information your application needed. As you move on and develop more complex custom WebCenter applications, you may want to connect to other databases for various content, and so on. You can use the same methodology to create a connection to your other databases.
- Install the WebCenter schema, which allowed you to use the Tags service. Having this schema available will now let you use both the Tags and Links services, which you can learn more about in the Oracle Fusion Middleware Developer's Guide for Oracle WebCenter.
- Create a simple custom WebCenter application, which allowed you to check out how to use the built-in WebCenter application template to create a basic JSF application.
- Create a customizable page, which took just a few steps to create using the Quick Start layout and a few customizable components from Oracle Composer. You also learned about the Component Palette, which contains a variety of ADF Faces components, ADF Layout components, and Oracle Composer components that you can use to develop your pages and application.
- Use Oracle Composer, both in your development environment (by adding the customizable components to your page), and in your runtime environment (by adding components like a text box). At runtime, you were able to see how easy it is for an end user to customize her own page, including moving components around and adding new components.
- Add WebCenter services (Search, Documents, and Tags services) to a page. By adding the Document and Search services, you enabled your users to browse content from a single content repository (in this case, your file system), and search for a keyword across your application. You also learned how to add tags to documents in a document library, and a “tag cloud” to visualize user-defined tags right in your application.

- Build a standards-based Java (JSR 168) portlet, which you built and coded in just a few steps, and can now reuse with other applications or portals.
- Register and define an OmniPortlet, which you added to your application, then developed by using an out-of-the-box user-friendly wizard at runtime.
- Wire two portlets, which enabled you to create user interaction at runtime by having the user actions in one portlet (the standards-based JSR 168 portlet) drive the content in the second portlet (the OmniPortlet).

Figure 6–1 shows the custom WebCenter application you created in this Tutorial.

Figure 6–1 Tutorial Result



You should now have a basic working knowledge of the fundamentals of Oracle WebCenter Framework.

Moving On

You can learn more about designing your own custom WebCenter applications, including using Oracle Composer, WebCenter Web 2.0 Services, and portlets, in the Oracle Fusion Middleware Developer's Guide for Oracle WebCenter.

To learn more about what you can do at runtime, including using Oracle Composer to customize pages, and how the various components behave and can be configured at runtime, see the Oracle Fusion Middleware User's Guide for Oracle WebCenter.

You can find all Oracle WebCenter Suite documentation on the WebCenter Documentation page on the Oracle Technology Network, at <http://www.oracle.com/technology/products/webcenter/documentation.html>.

You can learn more about other features of Oracle WebCenter Suite, and view demonstrations and see examples of custom WebCenter applications, portlets, and services in action on the Oracle WebCenter Suite home page on the Oracle Technology Network at:

<http://www.oracle.com/technology/products/webcenter/index.html>.

Index

A

- Application Navigator
 - viewing generated application files, 3-3
- applications
 - adding images, 3-6
 - adding Oracle Composer, 3-25
 - creating WebCenter applications, 3-2
 - files produced for a portlet producer application, 5-14
 - testing using the Integrated WLS, 5-39
 - using Oracle Composer, 3-31

C

- connections
 - creating for a content repository, 4-6
 - creating for Documents service, 4-6
 - creating to a database, 4-18
- content integration
 - using the Documents service, 4-10
- content repository
 - creating a connection, 4-6
- custom WebCenter applications
 - adding images to the resources, 3-6
 - adding Oracle Composer, 3-25
 - creating, 3-2
 - using Oracle Composer, 3-31
- customizable components
 - using, 3-11
- customization
 - enabling runtime, 3-25
 - performing at runtime, 3-31
- customize mode
 - testing, 5-39

D

- database
 - creating a connection, 4-18
 - installing sample schema, 2-3
 - installing the WebCenter schema, 2-5
- database connections
 - creating, 4-18
- deployment profile
 - WAR file, 5-31

Documents service

- adding a task flow, 4-10
- creating a connection, 4-6
- using at runtime, 4-15
- using with Tags, 4-22

I

Integrated WLS

- testing applications, 5-39
- using the default connection, 5-33

J

- JavaServer Faces pages, see pages, 3-8
- JSF pages, see pages, 3-8
- JSF, see pages, 3-8
- JSR 168 portlets
 - building, 5-3, 5-7
 - deploying, 5-31
 - files generated for, 5-14
 - testing customize mode, 5-39
 - testing with an application, 5-38
 - wiring with OmniPortlet, 5-53

L

layout components

- enabling page customization, 3-11

O

OmniPortlet

- adding to a page, 5-43
- customizing, 5-46
- registering the producer, 5-40
- wiring with a JSR 168 portlet, 5-53

Oracle Composer

- adding, 3-25
- using, 3-31

Oracle Technology Network, vi

Oracle WebCenter Framework

- checking for the extension, 2-1
- installing, 2-1

OTN

- see Oracle Technology Network, vi

P

pages

- adding Oracle Composer, 3-25
- creating, 3-8
- customizing at runtime, 3-31
- enabling customization, 3-11, 3-25

PDK-Java portlets

- adding OmniPortlet to a page, 5-43
- registering, 5-40

portlet producer application

- files generated for, 5-14

portlet producers

- registering, 5-40
- registering a WSRP producer, 5-36
- registering the preconfigured portlet producer, 5-40

portlet providers

- see WAR file, 5-31

portlets

- adding OmniPortlet, 5-43
- building a JSR 168 portlet, 5-3, 5-7
- customizing OmniPortlet, 5-46
- deploying to Integrated WLS, 5-31
- deploying using the Integrated WLS, 5-33
- enabling interaction, 5-53
- exposing as a web service, 5-36
- files generated for, 5-14
- registering the producer, 5-40
- testing a JSR 168 portlet with an application, 5-38
- testing customize mode, 5-39
- testing interaction, 5-56
- testing using the Integrated WLS, 5-39
- wiring, 5-53

preconfigured portlet producer

- registering, 5-40

S

sample files

- adding the database schema, 2-3
- downloading, 2-2

sample schema

- installing, 2-3

Search service

- adding a task flow, 4-2
- using with Tags, 4-26

services

- about, 4-1
- adding the Document Library task flow, 4-10
- adding the Search Toolbar task flow, 4-2
- adding the Tags service task flows, 4-19
- using Documents and Tags together, 4-22
- using Tags and Search together, 4-26
- using the Documents service at runtime, 4-15
- using the Tags service at runtime, 4-21

standards-based portlets

- building a JSR 168 portlet, 5-7
- deploying, 5-31
- exposing as a web service, 5-36
- files generated for, 5-14

- testing customize mode, 5-39
- testing with an application, 5-38

T

Tags service

- adding a task flow, 4-19
- installing the WebCenter schema, 2-5
- using at runtime, 4-21
- using with Documents, 4-22
- using with Search, 4-26

task flows

- adding the Documents service, 4-10
- adding the Search service, 4-2
- adding the Tags service, 4-19

W

WAR file

- creating, 5-31

Web 2.0

- about services, 4-1

web archive

- see WAR file, 5-31

web services

- exposing portlets as, 5-36

WebCenter applications

- creating, 3-2
- using Oracle Composer, 3-31

WebCenter Framework

- installing, 2-1

WebCenter schema

- installing, 2-5

WebCenter Web 2.0 Services

- about, 4-1

WebLogic Server

- testing applications, 5-39
- using the default connection, 5-33

WSDL

- publishing a portlet as, 5-36

WSRP producers

- creating, 5-31
- registering, 5-36