Oracle® Fusion Middleware

Tutorial for Oracle WebCenter Developers

11*g* Release 1 (11.1.1)

E10273-03

December 2009

ORACLE®

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

E10273-03

Primary Author: Vanessa Wang

Contributor: Robin Fisher, Medini Kakade, Peter Moskovits, Kundan Vyas

Contents

5 Adding Oracle WebCenter Services to Your Application

6 Building Portlets and Wiring Them in Your Application

7 Changing the Look and Feel of Your Application

8 Conclusion

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.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.

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

1

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.

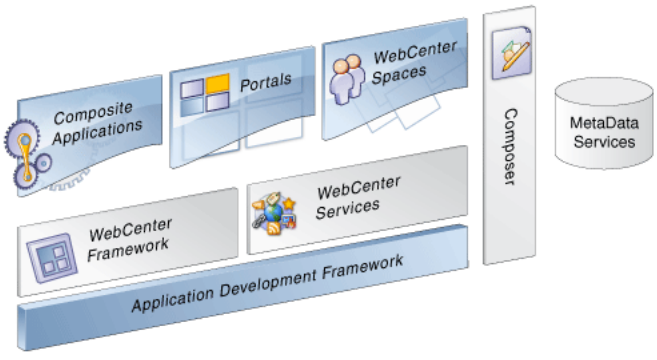
If you need additional information about a feature, you can always refer to the *Oracle Fusion Middleware Developer's Guide for Oracle WebCenter* and the *Oracle Fusion Middleware User's Guide for Oracle WebCenter*.

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 method for layering 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.

Figure 1–1 Overview of the Oracle WebCenter Architecture



In Figure 1–1, notice WebCenter Services and Composer (or, Oracle Composer). You will use both of these components in conjunction with WebCenter Framework in this Tutorial.

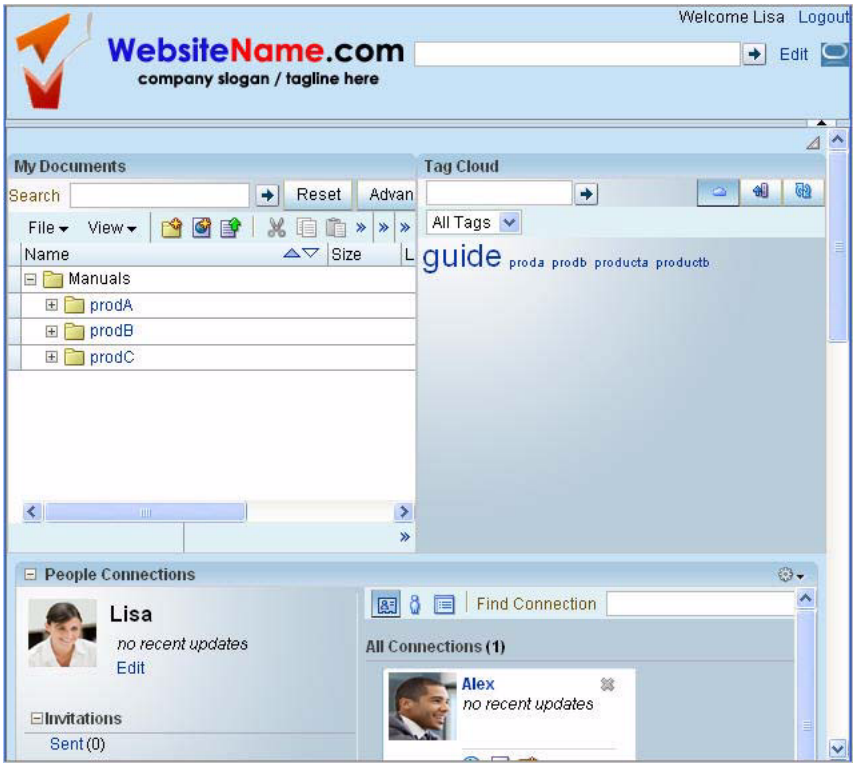
For more information about WebCenter Framework, WebCenter Services, and Composer, refer to Chapter 1, "Understanding Oracle WebCenter"in the *Oracle Fusion Middleware Developer's Guide for Oracle WebCenter*.

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. You will enable interaction between the two portlets, so that user actions performed on one portlet drives the content that displays in the second portlet. Finally, you will change the look and feel of the application using a skin.

Figure 1–2 shows a partial view of custom WebCenter application you will create in this Tutorial. To see a quick preview of what you will build at the beginning of each lesson, refer to the chapters themselves.

Figure 1–2 Partial View of the Application You Will Create



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, adding resources to your 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 Security to Your Application" tells you how to implement basic ADF security in an application, create users and roles, then assign ADF Security Policies to a page.
- Chapter 5, "Adding Oracle WebCenter 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, create a social network, add tagging and a tag cloud to your application, add and use links, and, optionally, integrate email. You will also learn how to use each of these services at runtime.
- Chapter 6, "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.
- Chapter 7, "Changing the Look and Feel of Your Application" shows you how to change the look and feel of your application using a skin. You will also learn how to use Oracle Composer at runtime to personalize the application as different user types, based on the users created in Chapter 4, "Adding Security to Your Application."

2

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 will install the database schemas.

Introduction

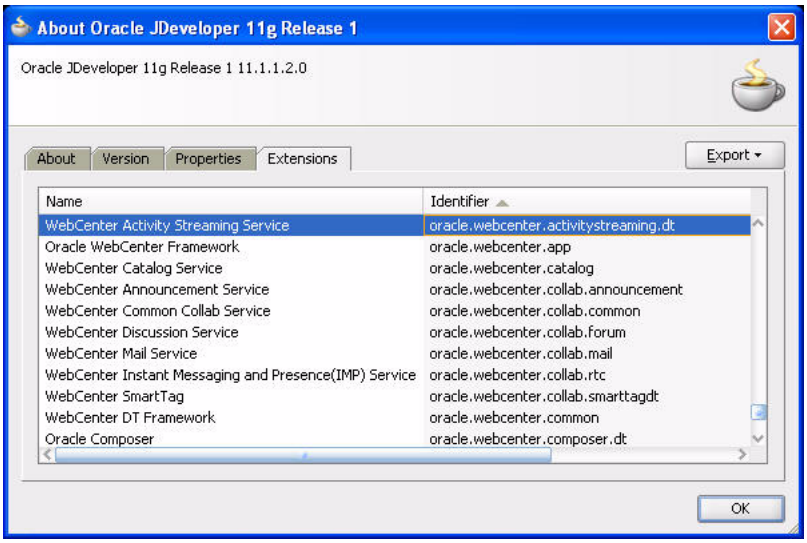
We will set up the environment for the Tutorial by following these steps:

- 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 11*g* 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 **About** from the Help menu, then click the **Extensions** tab. At the top of the About dialog, you should see **Oracle JDeveloper 11g Release 1 11.1.1.2.0**. 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 displays, select **Default Role** to enable all technologies, and click **OK**.
3. If a dialog displays asking if you want to migrate settings from an earlier version, click **No**.
4. In Oracle JDeveloper, choose **Check for Updates** from the Help menu.
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:

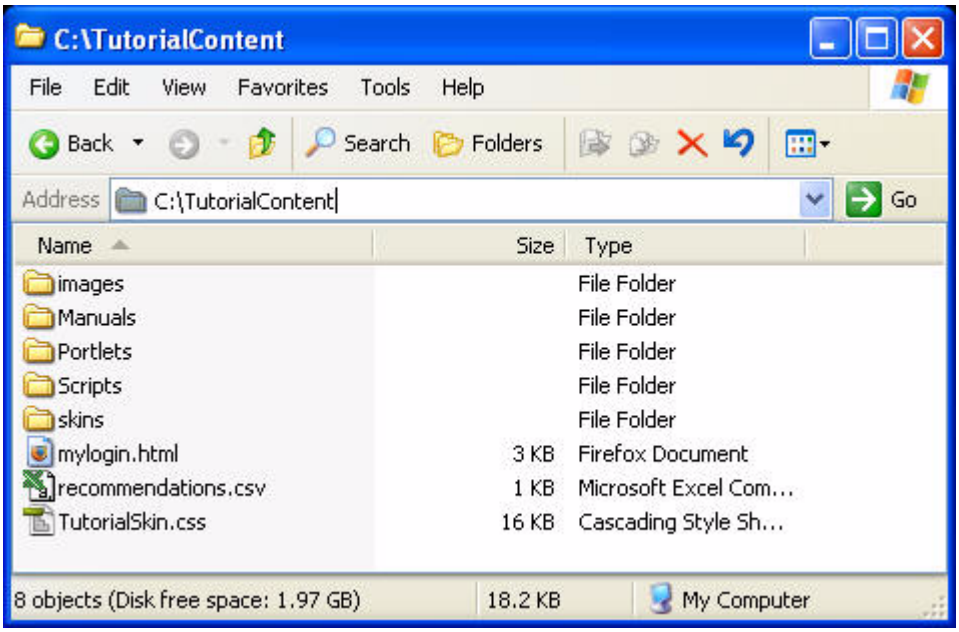
1. Open a browser, and enter the following in the Address field:

   `http://www.oracle.com/technology/products/webcenter/files/webcentertutorialcontent_11120.zip`

2. Open the ZIP file (`webcentertutorialcontent_11120.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.

Note: If you see an error that says:

```
DROP USER FOD CASCADE
*
ERROR at line 1:
ORA-01918: user 'FOD' does not exist,
```

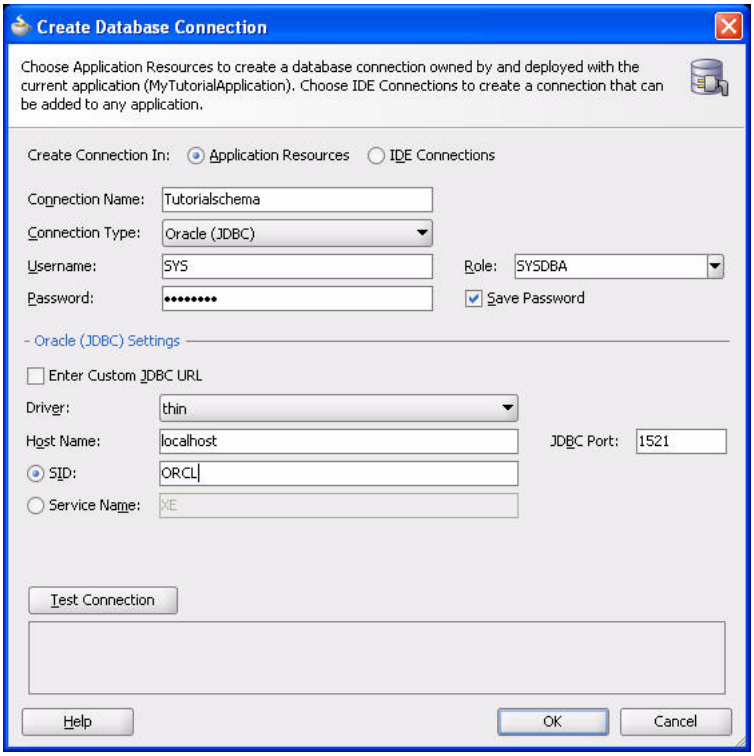
you can ignore this message, as it means that the schema does not yet exist.

To add the sample schema to your database:

1. In Oracle JDeveloper, from the **Tools** menu, choose **Database**, then choose **SQL Worksheet**.

2. In the Create Database Connection dialog, 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 the appropriate role from the Role list (for example, `SYSDBA` or `SYSOPER`; if using the `SYSTEM` user or a user that does not need the `SYSDBA` or `SYSOPER` role, then leave this field blank)
   - **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, then click **OK** again to close the Select Connection dialog. The SQL Worksheet should display. Otherwise, choose **SQL Worksheet** from the Tools menu, then select the newly created connection and click **OK**.
4. In the SQL Worksheet panel, create the schema by entering the following command (at the bottom of the page that displays, you will see that the tab is called "SQL Worksheet"):

```
create user fod identified by fusion;
```

5. Ensure your cursor is in the SQL statement you just entered, then click the **Execute Statement** icon at the top of the panel.
6. 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 provided to access the schema in your database.
7. Click the **Execute Statement** icon (the green arrow) at the top of the panel.

Figure 2–4 Execute Statement Icon

8. From the **Tools** menu, choose **Database**, then choose **SQL Worksheet**.
9. In the Select Connection dialog, click the pencil icon to edit the connection.
10. 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.
11. Click **OK** to close the Select Connection dialog.
12. Close the **SQL Worksheet** panel.
13. Start a new SQL Worksheet using the new connection. Choose **SQL Worksheet** from the Tools menu again.
14. Create the schema objects by executing the `buildFromJDev.sql` script that is 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 provided script, `c:\TutorialContent\Scripts\build.sql`.

Step 4: Install the WebCenter Schema

To use the Tag, Links, and People Connections services, 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:

1. From the **Tools** menu, select **SQL Worksheet**.
2. In the Select Connection dialog, click the pencil icon to edit the connection.
3. Modify the connection to use an administrator username and password, such as `SYS` (using the `SYSDBA` role) then click **OK**.
4. Click **OK** to close the Select Connection dialog.

5. From the Tools menu, choose **SQL Worksheet**.
6. Enter the following SQL statement in the SQL Worksheet panel:

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

   where `JDEV_HOME` is the location where JDeveloper is installed on your machine.
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 Tablespaces, re-enter the default values `users` and `temp`, then accept them.

Now that you have set up the files and the database for your environment, you are ready to begin!

3

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 update your application with existing resources (like images and a CSS file) that we have provided so that you can use them with your application. 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 the page you will create in 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 Resource 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.

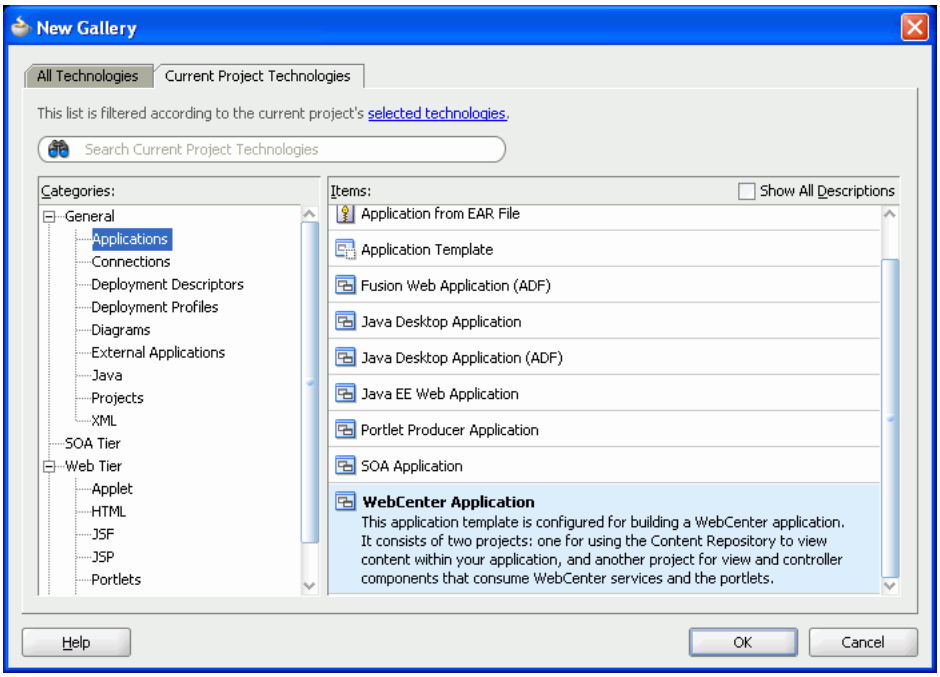
Step 1: Create a Custom WebCenter Application

Let's begin by building a simple custom WebCenter application. You will then create a page for this application that will serve as the interface for the application. This step introduces you to the WebCenter application template, as well as the Application Navigator, which helps you view various parts of your application.

To create a custom WebCenter application:

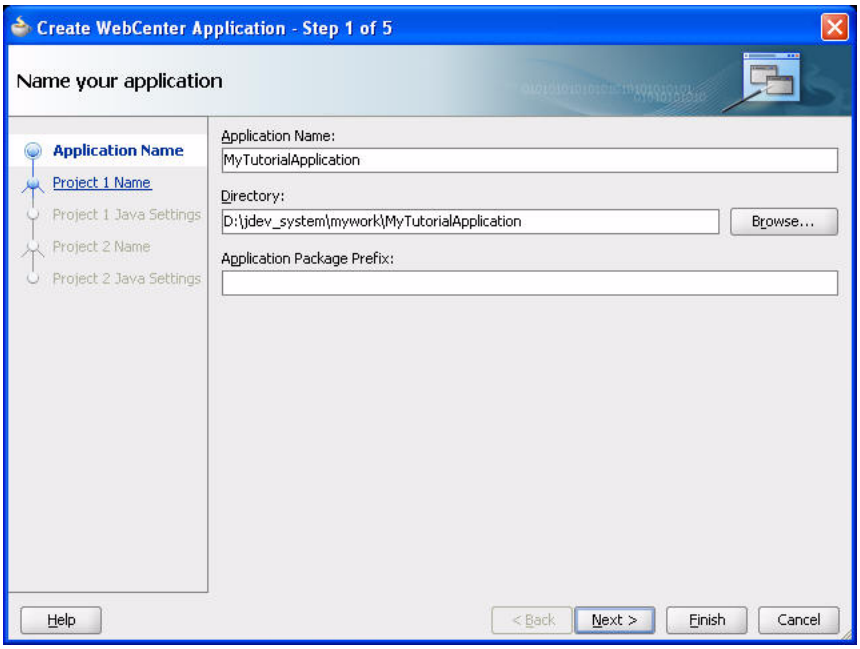
1. In Oracle JDeveloper, from the File menu, choose **New**.
2. In the New Gallery, on the Current Project Technologies tab, you should see the General category highlighted. Under General, click **Applications**.
3. In the Items list, scroll down and select **WebCenter Application**, then click **OK**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

Figure 3–2 Create New WebCenter Application



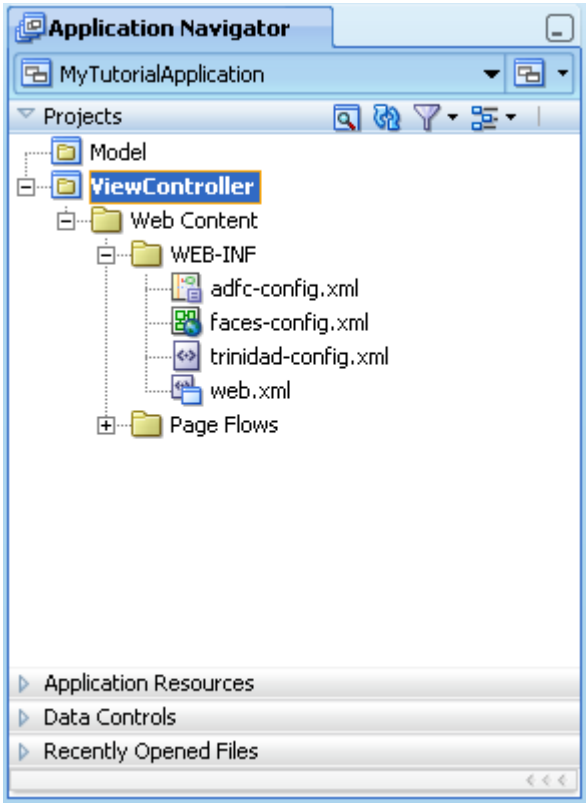
4. On the Application Name tab, in the Application Name field, enter `MyTutorialApplication` and leave the other default values as they are, as shown in Figure 3–3. In this example, the `JDEV_USER_HOME` is `d:\jdev_system`. However, your directory may be different.

Figure 3–3 Name a New WebCenter Application



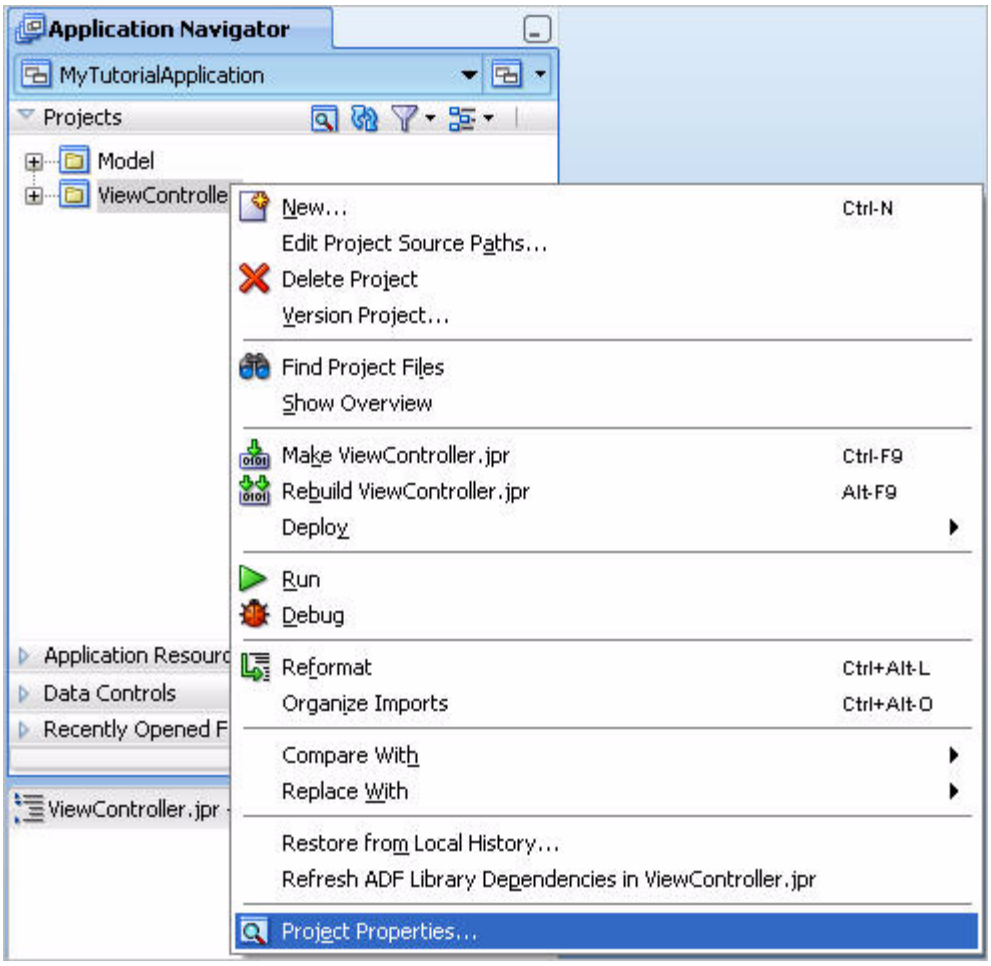
5. Click **Finish**. Oracle JDeveloper generates the base files, including two projects, for your application, which you can see in the Application Navigator (Figure 3–4):
   - **Model**, in which you define the JavaBeans and other data controls you need if the application is to perform any back-end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

Figure 3–4 Generated Application Project Files in the Application Navigator



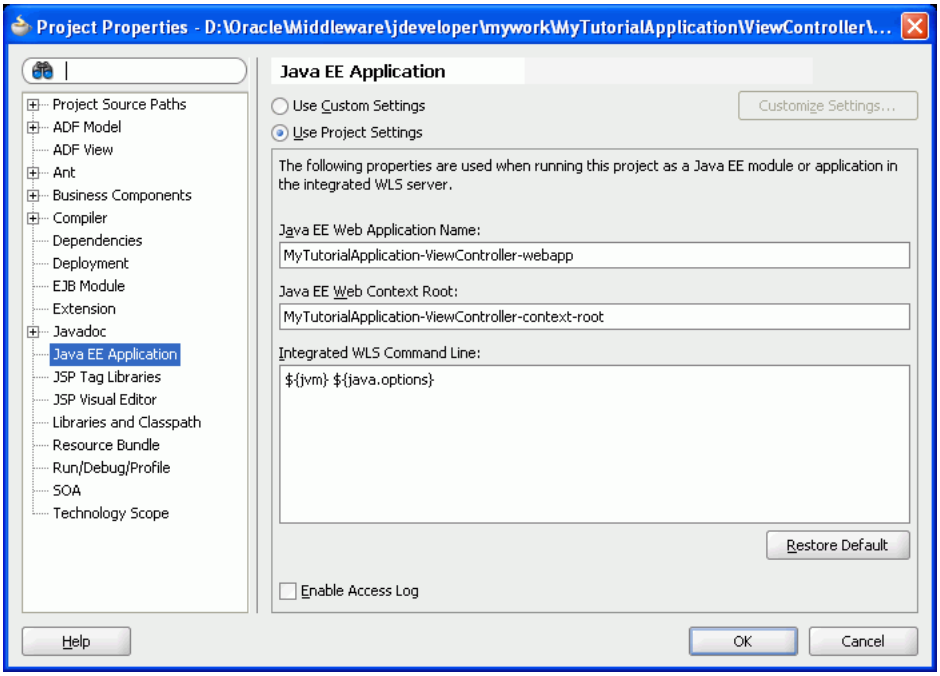
6. Let us make a few adjustments for the purposes of testing our application. Right-click the **ViewController** project, then choose **Project Properties** (Figure 3–5).

Figure 3–5 Editing the Project Properties



7. In the Project Properties dialog, in the left column, choose **Java EE Application** (Figure 3–6).

Figure 3–6 Project Properties: Java EE Application



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

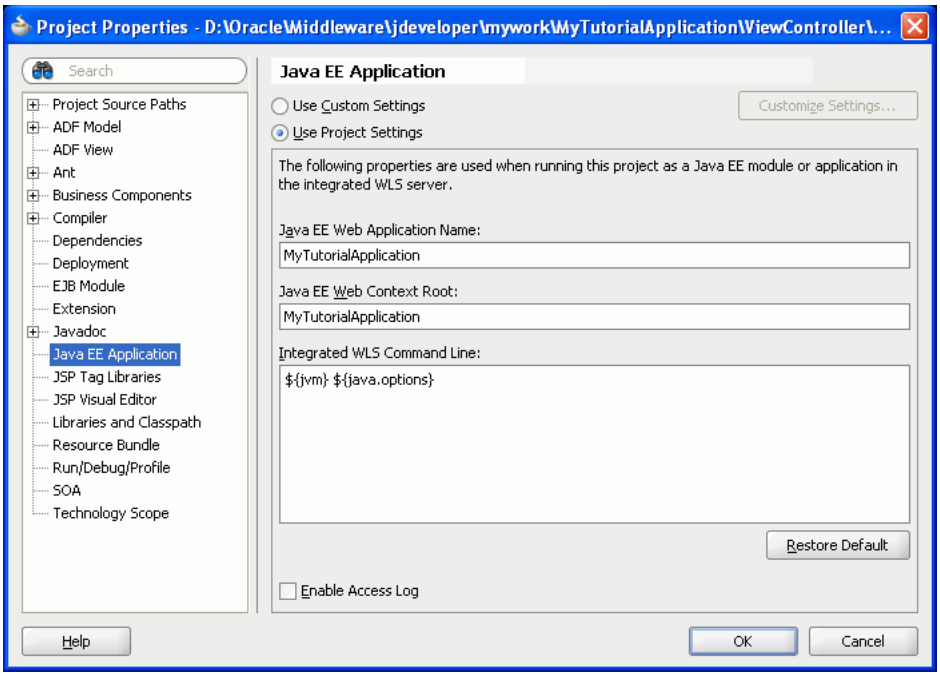
   `MyTutorialApplication`

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

   `MyTutorialApplication`

   The Project Properties dialog should now look like Figure 3–7:

Figure 3–7 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.

For more information on creating an application based on the WebCenter template, see Chapter 3, "Preparing Your Development Environment" in *Oracle Fusion Middleware Developer's Guide for Oracle WebCenter*.

Now that you have created a basic custom WebCenter application, you can create a page. Before we create the page, though, we must add the resource files to our application so that we can use them with our page.

Step 2: Add the Resource Files to the Application

Now that you have created the application, you can add the resource files that you will need to complete the steps in the Tutorial. Here, you will add the image files, the CSS file, and the login page that you downloaded in Chapter 2, "Preparing for the Tutorial." This step shows you how you can take *existing* files (for example, your own organization's logo images) and incorporate them into a WebCenter application.

To add the resource files to the application:

1. Ensure that you have followed the steps in Chapter 2, "Preparing for the Tutorial," which includes the step for obtaining the Tutorial sample files you will add.
2. 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:

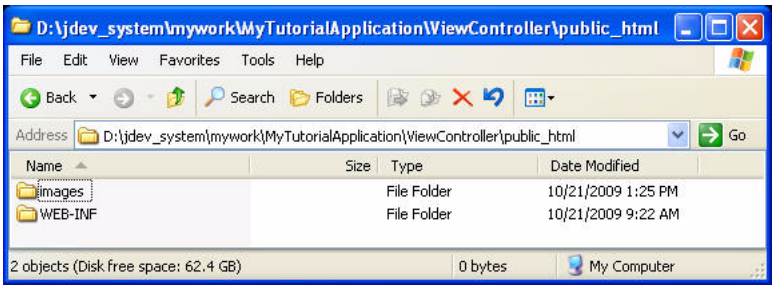
   `JDEV_USER_HOME``\mywork\MyTutorialApplication\ViewController\public_html`

   `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.

Figure 3–8 shows an example of where the images folder will be located where the `JDEV_USER_HOME` is set to `d:\jdev_system`.

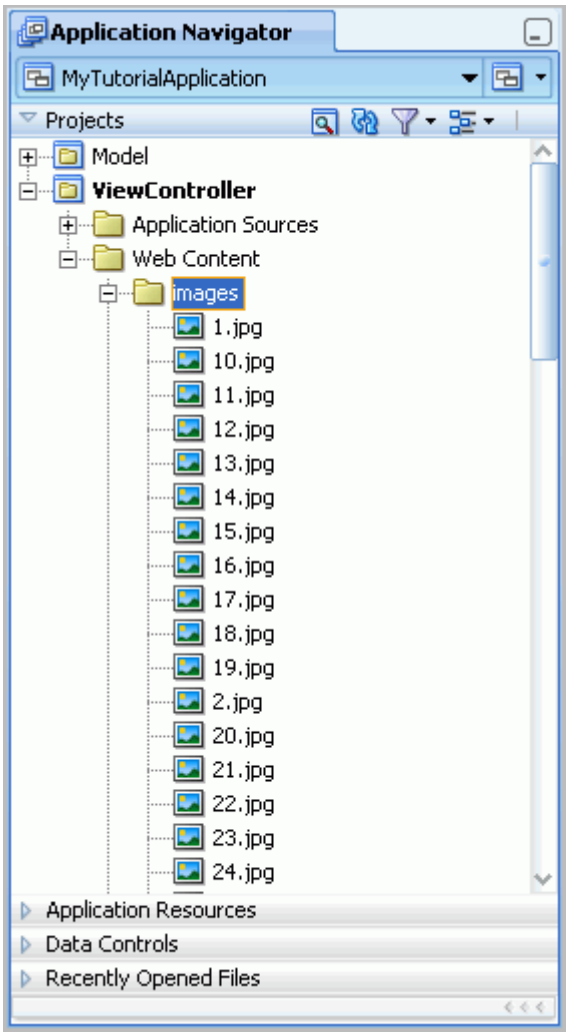
Figure 3–8 Example of the Images folder in the JDEV_USER_HOME



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

3. Locate the Tutorial sample files you downloaded and extracted in Chapter 2, "Preparing for the Tutorial," and copy the `C:\TutorialContent\images` folder to your *`JDEV_USER_HOME`*`\mywork\MyTutorialApplication\ViewController\public_html` 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 3–9).

Figure 3–9 Images in the Application Navigator



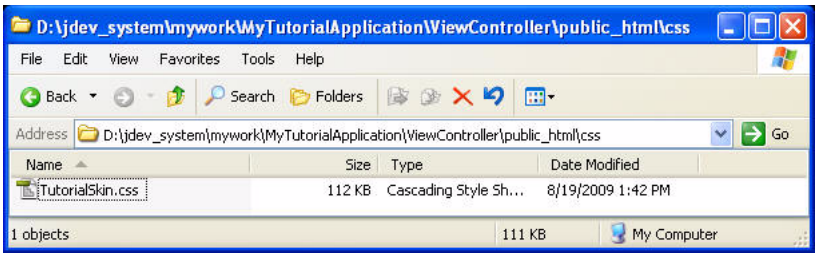
Note: Notice that the image names are enumerated: `1.jpg`, `2.jpg`, and so on. These image names match the product ID of the items we will retrieve when we create our portlets in Chapter 6, "Building Portlets and Wiring Them in Your Application." If you change the names of these image files, the statement may not return the correct results.

You can now use these images with your application.

While you are here, also add the CSS skin and its associated images that you will apply to the application in Chapter 7, "Step 1: Change the Application Look and Feel Using a Skin". You will not do anything with this skin just yet.

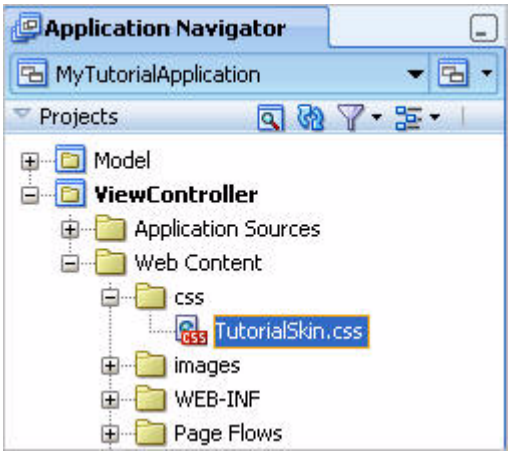
5. Return to your file system.
6. In your file system explorer, copy the `C:\TutorialContent\css` folder to your *`JDEV_USER_HOME`*`\mywork\MyTutorialApplication\ViewController\public_html` directory (Figure 3–10).

Figure 3–10 CSS Directory with the TutorialSkin.CSS file in the JDEV_USER_HOME



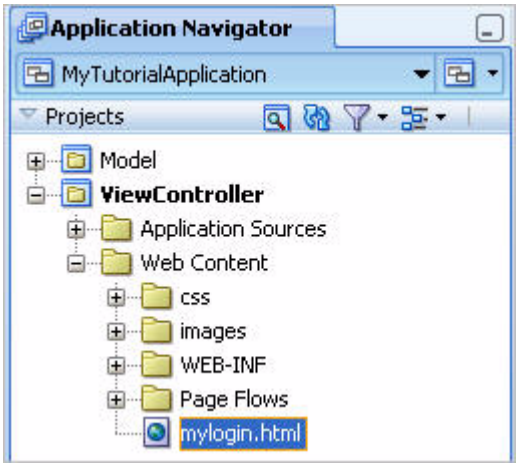
7. Return to JDeveloper and refresh the Application Navigator to see the css directory and its contents (`TutorialSkin.css`), as shown in Figure 3–11.

Figure 3–11 CSS File in the Application Navigator



8. Next, add the new `mylogin.html` page that you will use in Chapter 4, "Adding Security to Your Application." In the files that you downloaded, `C:\TutorialContent`, locate `mylogin.html`.
9. Copy and paste this file to the `public_html` folder.
10. Return to JDeveloper and refresh the Application Navigator. You should see the **mylogin.html** file in the **Web Content** directory, as shown in Figure 3–12.

Figure 3–12 Mylogin.html Page in the Web Content Directory



You now have all the resource files you need to complete the steps in this Tutorial.

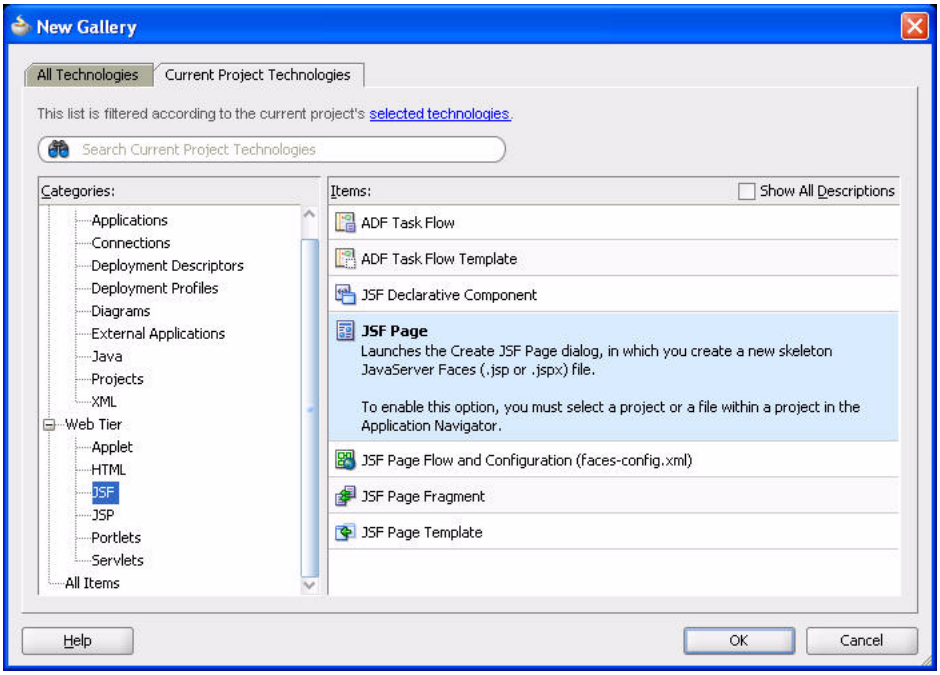
Step 3: Create a Page

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 will configure and add in the subsequent steps of this Tutorial. You will use one of Oracle WebCenter Framework's pre-built "quick start" layouts as a starting point. This step introduces you to the Create JSF Page dialog and the Visual Editor, as well as the Structure window.

To create a page:

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 3–13).

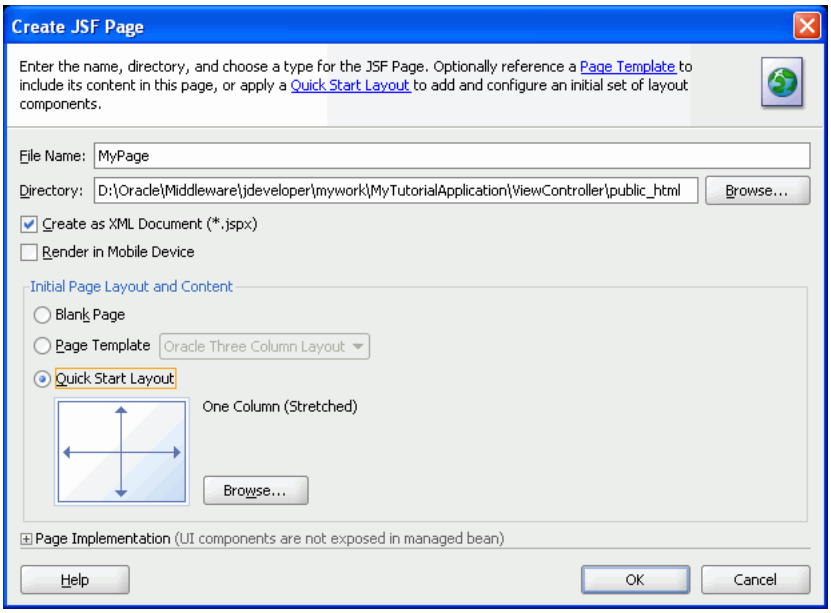
Figure 3–13 Choosing JSF Page from the New Gallery



4. In the Create JSF Page dialog, in the Name field, enter `MyPage`.
5. Select the **Create as XML Document (*.jspx)** checkbox.
6. 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, you will use a quick start layout.

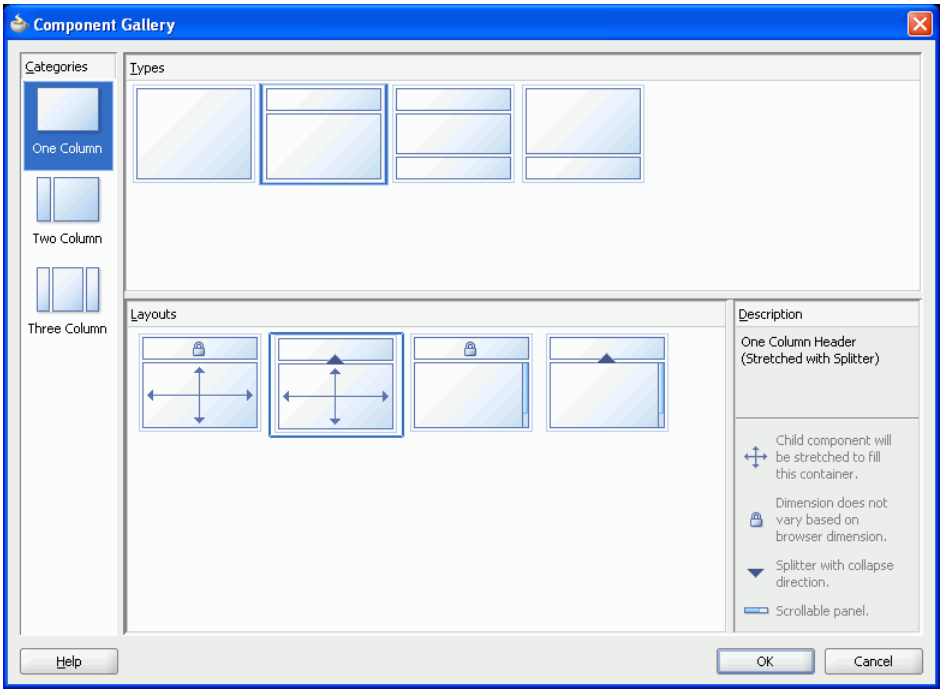
   Under Initial Page Layout and Content, select **Quick Start Layout**, then click **Browse** (Figure 3–14).

Figure 3–14 Create JSF Page Dialog



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 3–15.

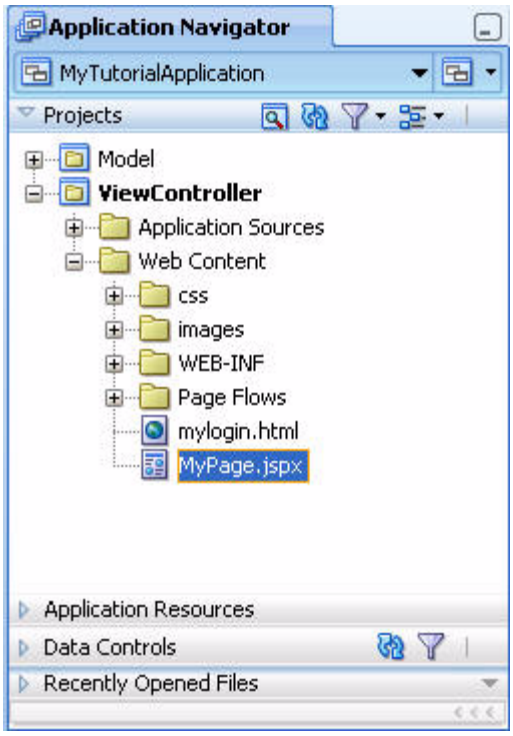
Figure 3–15 Selecting an Initial Layout



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

    JDeveloper finishes the wizard and displays your page in the Application Navigator (Figure 3–16).

Figure 3–16 MyPage in the Application Navigator



11. To the right of the Application Navigator, notice that your page displays in the Visual Editor (the large area between the Application Navigator and the Component and Resource Palettes) in the Design view (notice the Design tab is highlighted at the bottom of the view), as shown in Figure 3–17.

Figure 3–17 Design tab

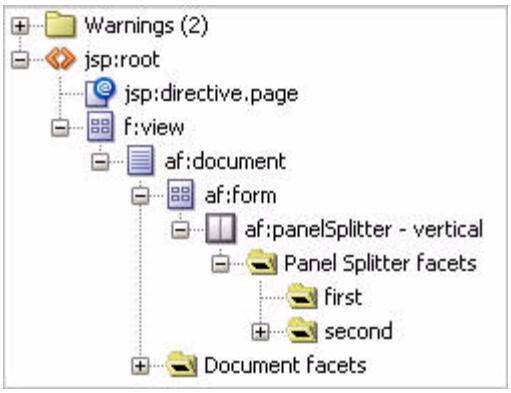
12. Below the Application Navigator, notice the Structure window for your page. This view shows all the elements of your page in a hierarchical view. Just below that Structure tab, notice the **pushpin** icon (Figure 3–18).

Figure 3–18 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 3–19).

Figure 3–19 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.

For more information on creating pages using Oracle WebCenter Framework, see Chapter 3, "Preparing Your Development Environment" in *Oracle Fusion Middleware Developer's Guide for Oracle WebCenter*.

Now that we have created our application and a JSF page, we can 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

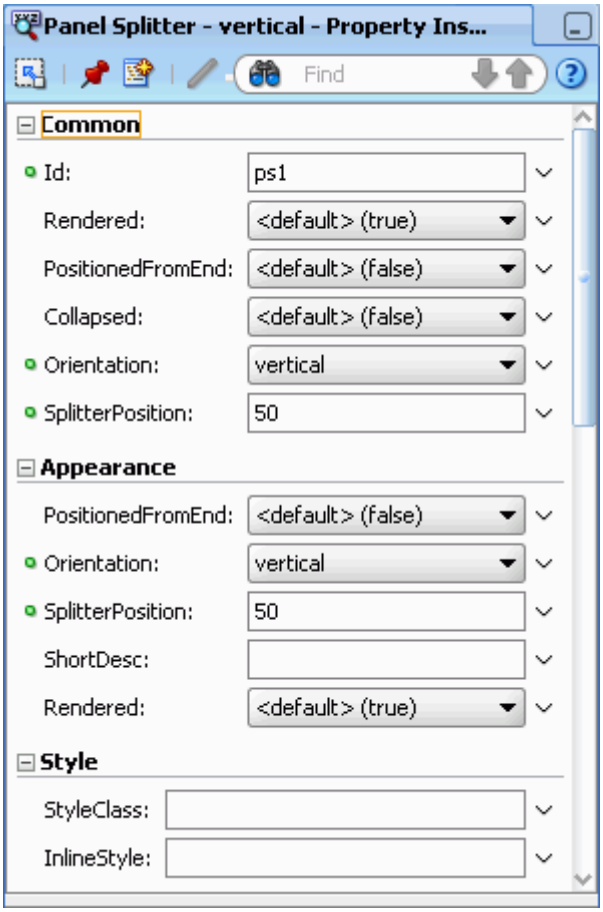
Selecting the Quick Start Layout when we created the page gave us a few "starter" layout components, specifically the Panel Splitter, which provides the basic framework of your page. In this step, you will enhance the layout and create a structure for the header section where you can add content, such as a corporate logo. You will learn about using the Structure window to tightly control the layout components on your page, as well as using components on the Component Palette.

To add layout components 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 3–20). 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.

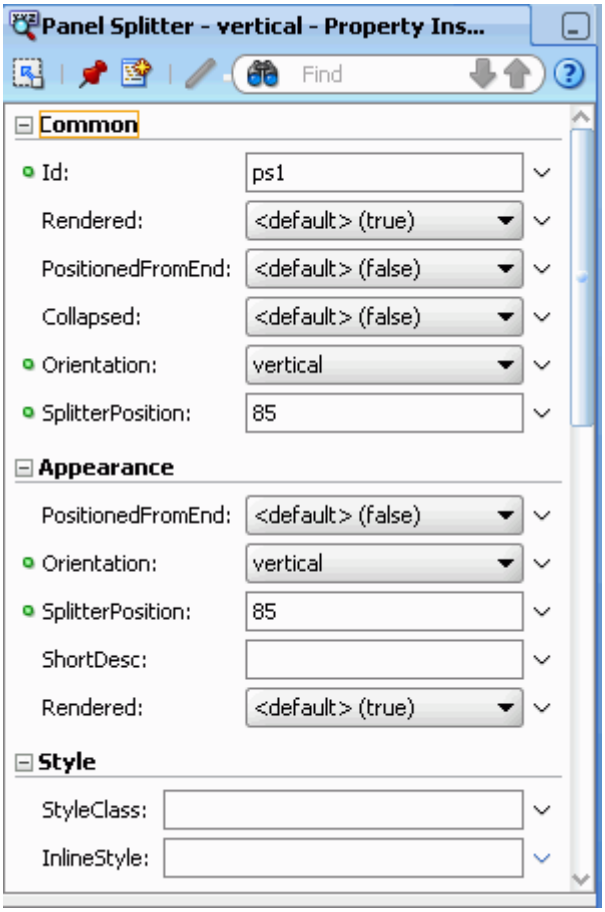
   Note: You can always view the Property Inspector for a component by right-clicking the component, then choosing **Go To Properties** from the context menu.

Figure 3–20 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 3–21). Clicking elsewhere in JDeveloper sets the property.

Figure 3–21 Setting the SplitterPosition Property



3. In the Structure window, expand the **Panel Splitter**. Notice that it contains two facets. In the next step, we will 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 **Component Palette** from the View menu.

   > **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 3–22), and drag and drop it onto the `first` facet in the Structure window.

Figure 3–22 Panel Stretch Layout in the Component Palette

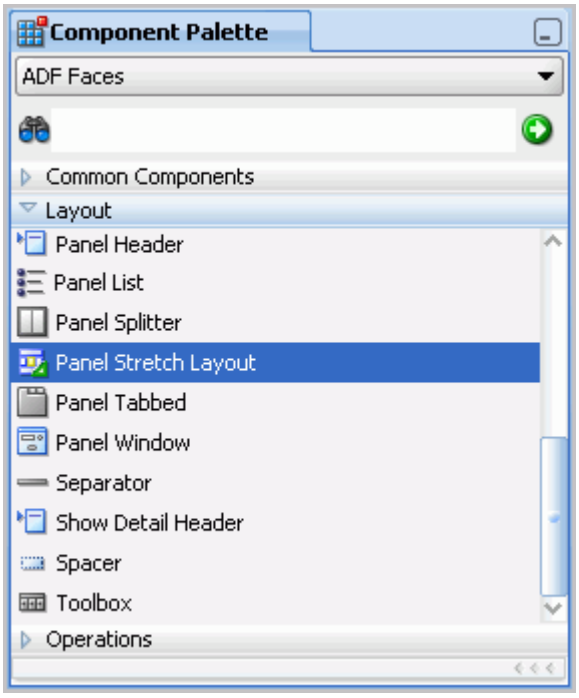
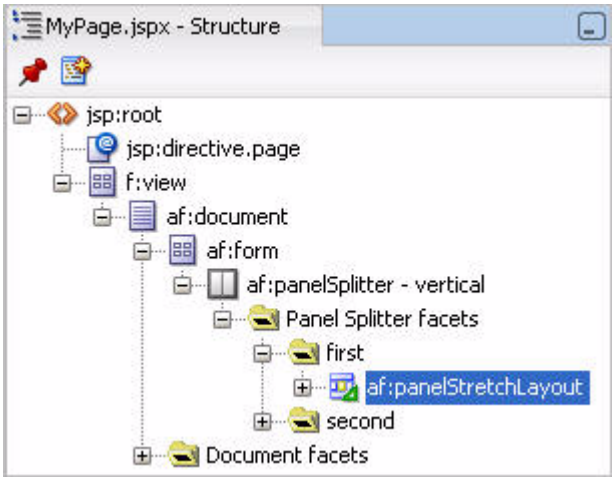


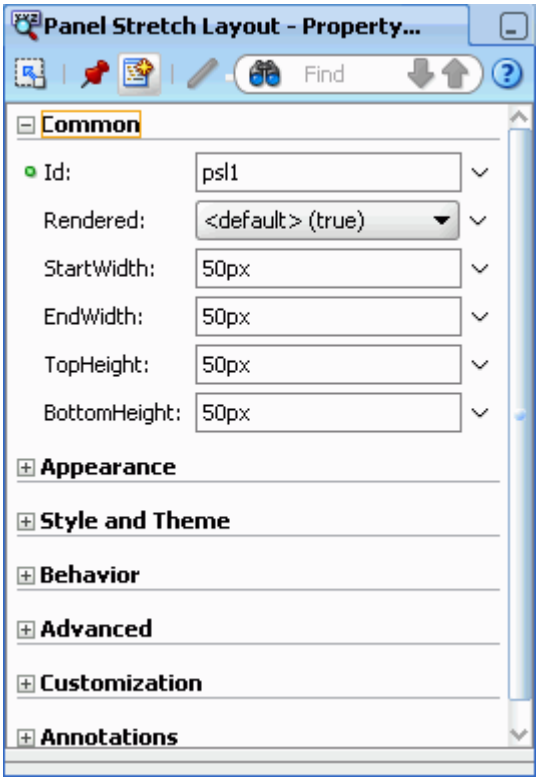
Figure 3–23 shows the Panel Stretch Layout in the Structure window.

Figure 3–23 Panel Stretch Layout in the Structure Window



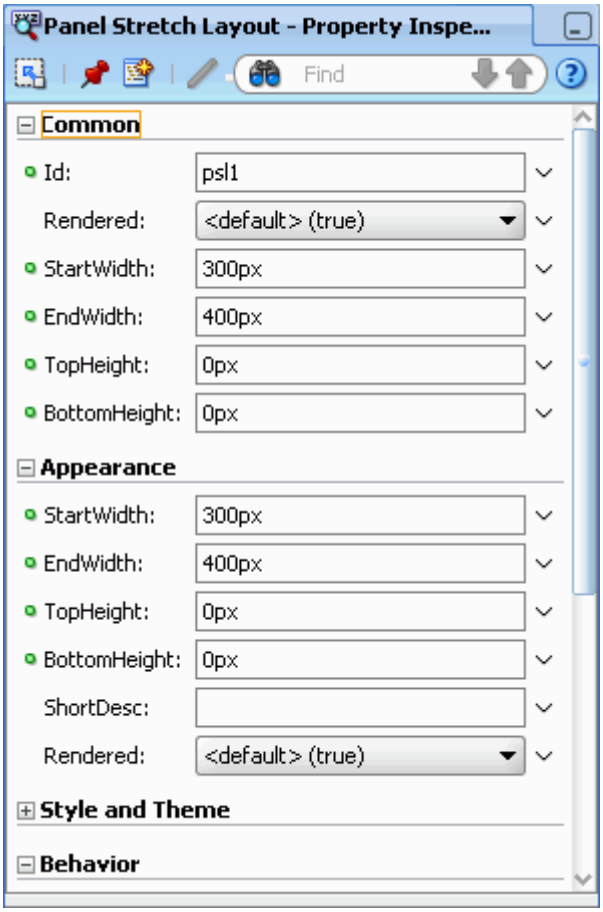
6. When you drop the Panel Stretch Layout component, you'll notice that the Property Inspector now displays the properties for this component(Figure 3–24).

Figure 3–24 Properties of the Panel Stretch Layout



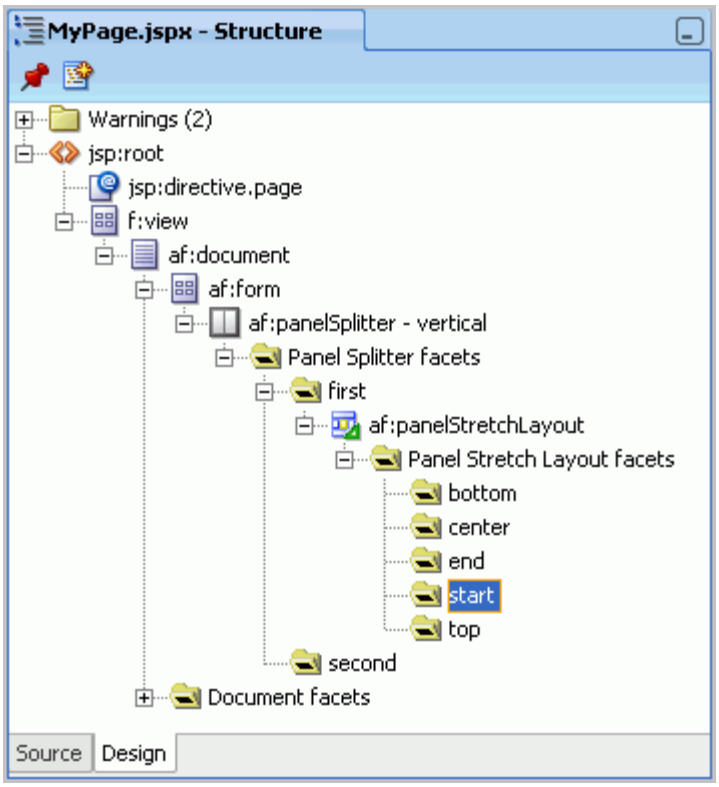
7. Set the following properties in the Property Inspector, as shown in Figure 3–25:
   - **StartWidth**: `300px`
   - **EndWidth**: `400px`
   - **TopHeight**: `0px`
   - **Bottomheight**: `0px`

Figure 3–25 Property Inspector for the Panel Stretch Layout with Updated Values



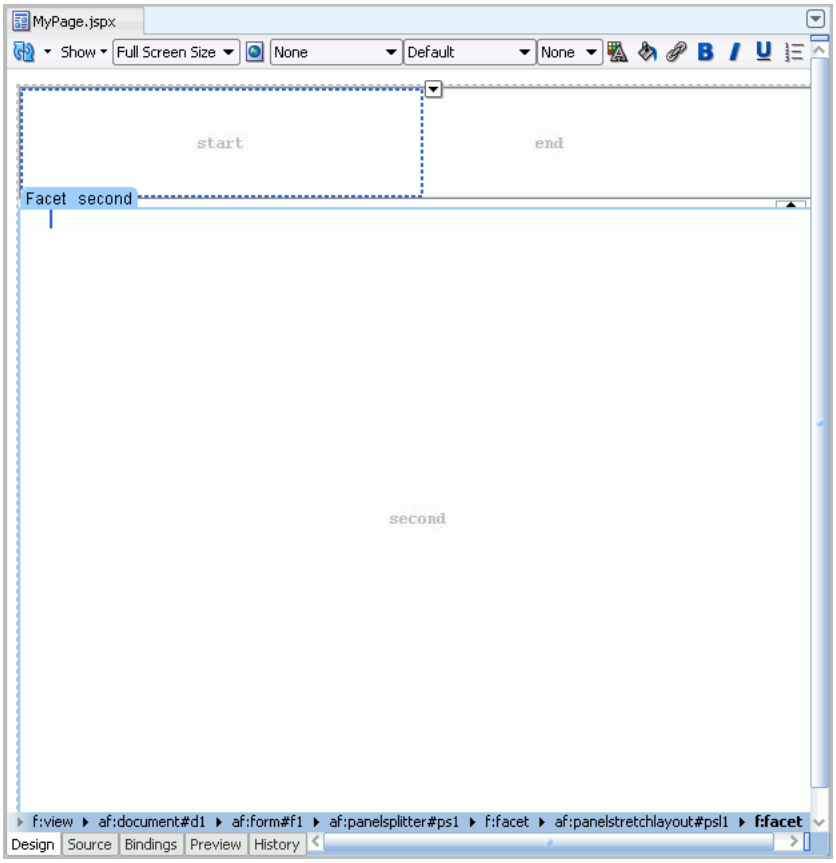
8. Next, add a logo to the header of the page, in the `start` facet of the Panel Stretch Layout. In the Structure window, expand the Panel Stretch Layout you just added so that you can see the different facets (Figure 3–26), then select the `start` facet.

Figure 3–26 Panel Stretch Layout Facets



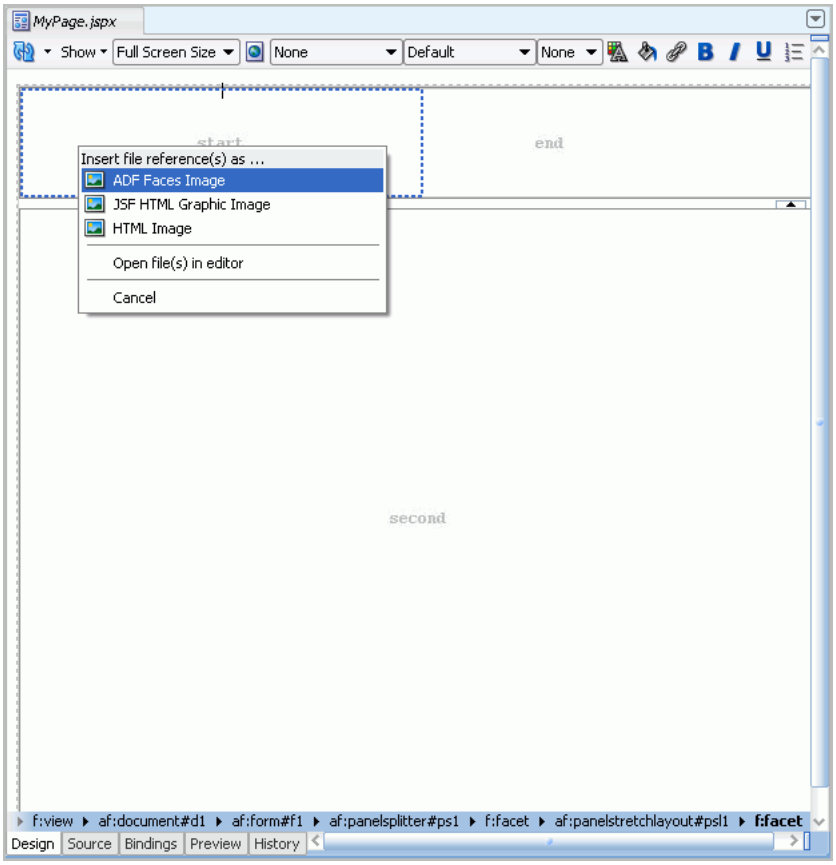
Notice how the corresponding facet is highlighted in the Design view (Figure 3–27). This is where we will add the logo image.

Figure 3–27 Start Facet in the Design View



9. The logo we will add is an image file we added to our application in Step 2: Add the Resource Files to the Application. In the Application Navigator, under **ViewController**, then **Web Content**, open the **images** folder.
10. Locate the `logo.png` file.
11. 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 3–28).

Figure 3–28 Choosing ADF Faces Image from the Context Menu

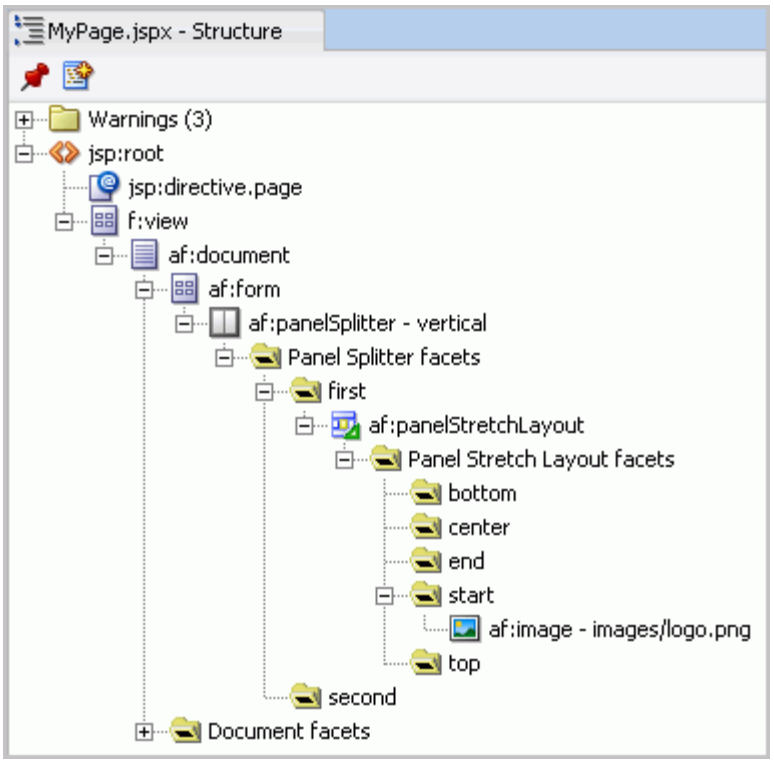


The logo displays on your page, as shown in Figure 3–29.

Figure 3–29 Logo Image on Page in the Start Facet

Also notice that the image now displays in the Structure window, as shown in Figure 3–30.

Figure 3–30 Logo Image in the Structure Window

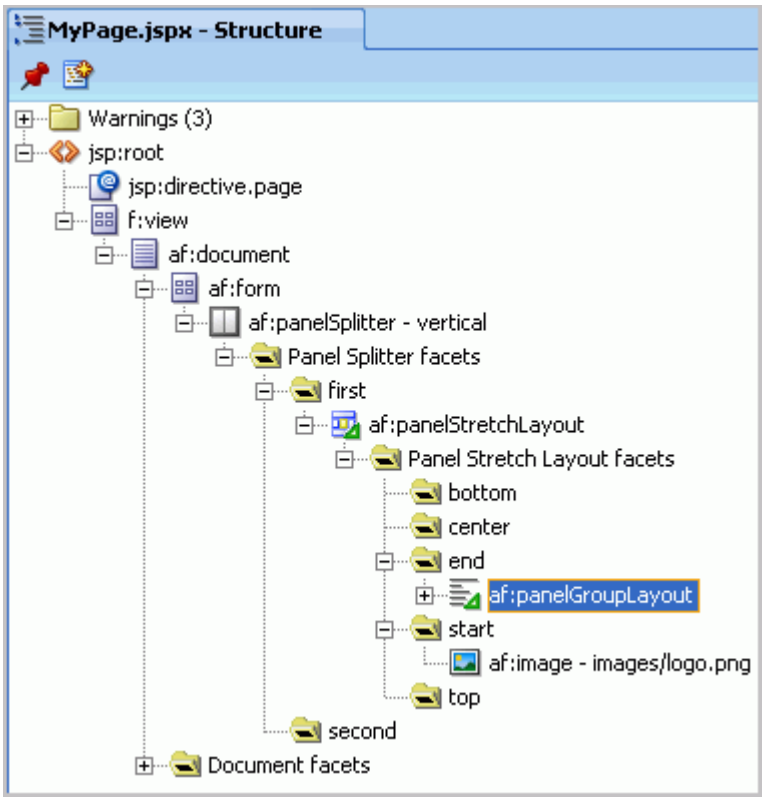


12. Now, you can finish the layout of the header. In the Design view, notice the `end` facet (Figure 3–31) that displays to the right of the logo.

Figure 3–31 End Facet in the Design View

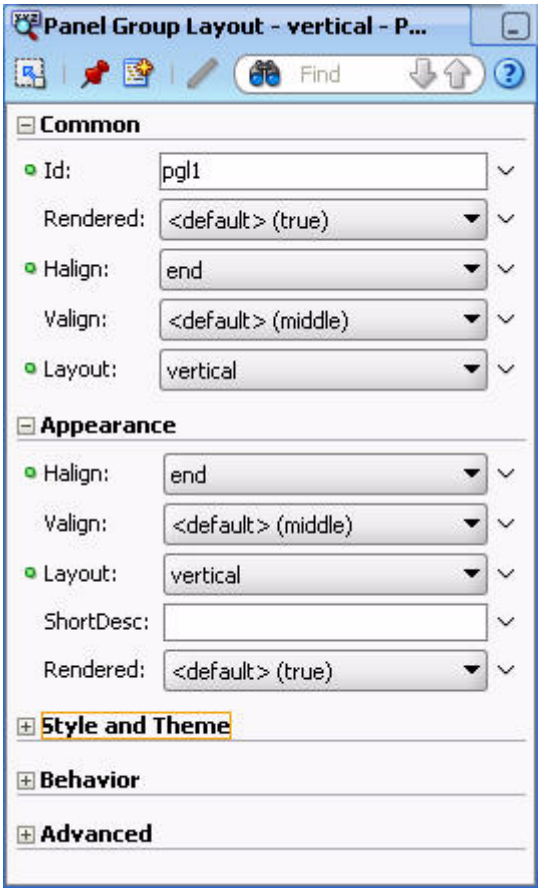
From the Component Palette, drag and drop the **Panel Group Layout** ADF Faces component onto this facet. You can see this now either in the Design view or, more easily in the Structure window (Figure 3–32).

Figure 3–32 Panel Group Layout in the End Facet



13. 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.
14. Change the **Layout** property to **vertical** (Figure 3–33).

Figure 3–33 Changing the Properties of the Panel Group Layout



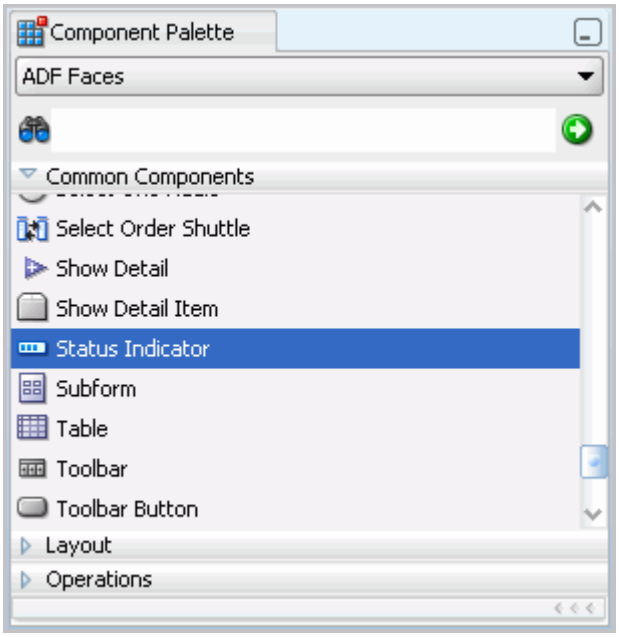
15. For the purposes of this Tutorial, let's add a Status Indicator. The Status Indicator keeps us informed of the application's activity during runtime. For example, if you click a link, the Status Indicator will let you know the application is accessing the target of that link.

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

16. In the Component Palette, choose **ADF Faces** from the list to display the ADF Faces components.

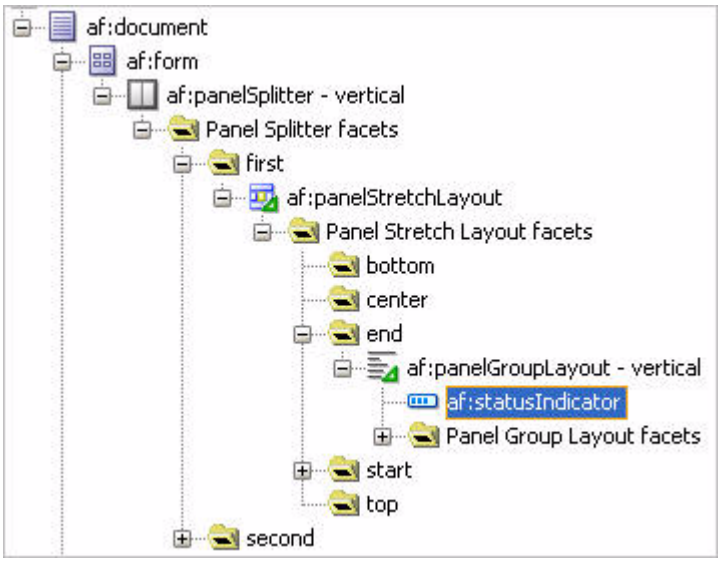
17. Under Common Components, scroll down the list and locate **Status Indicator**, then select it (Figure 3–34).

Figure 3–34 Status Indicator in the Component Palette



18. Drag and drop the **Status Indicator** onto the Panel Group Layout in the `end` facet, as shown in Figure 3–35.

Figure 3–35 Status Indicator on the Panel Group Layout



19. Let us 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 3–36.

Figure 3–36 MyPage with Logo and Status Indicator at Runtime

20. Return to JDeveloper.

    Now that we have set up the initial header for our page, we will 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.

In this step, we will add a few features that Oracle Composer offers. For a more in-depth description of Oracle Composer, however, see the chapters in Part II, "Using, Extending, and Customizing Your Application with Oracle Composer" in *Oracle Fusion Middleware Developer's Guide for Oracle WebCenter*.

To add Oracle Composer to the page:

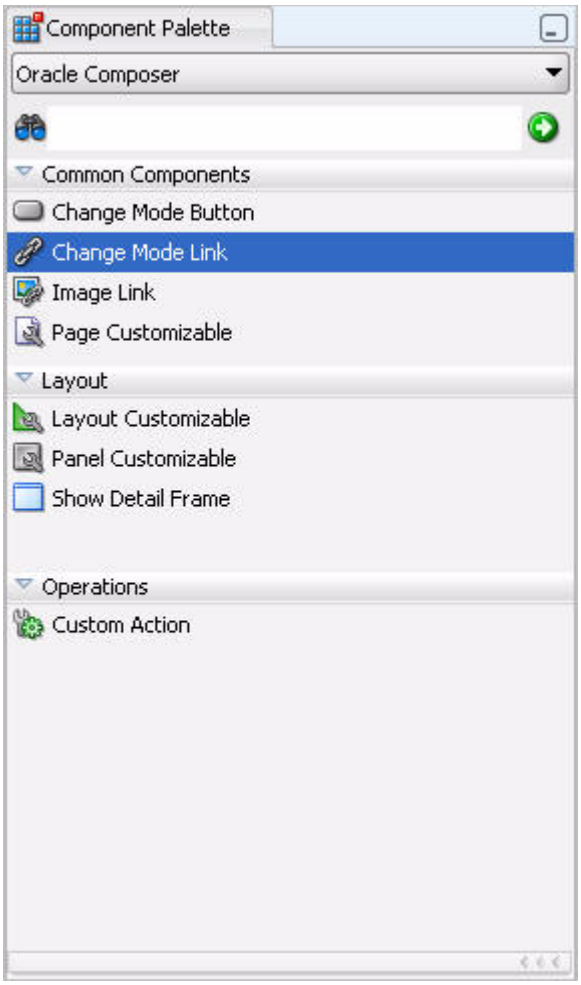
1. First, add a Change Mode Link to the page. This link will enable runtime users of your application to switch between viewing the page and editing it using Oracle Composer.

   In the Component Palette, choose **Oracle Composer** from the list at the top. In the previous step, you chose ADF Faces from this list, but let's switch views so we can see Oracle Composer's components.

2. Under Common Components, drag and drop the **Change Mode Link** 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 3–37 shows the Change Mode Link in the Component Palette.

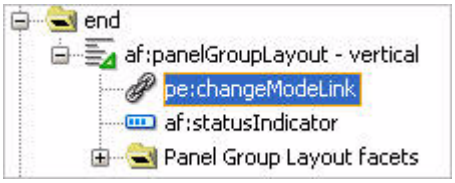
   Note: You can adjust the display of the Component Palette to see all of the options. You can do so by clicking and dragging the bars (such as the Layout bar) up and down to view the component names.

Figure 3–37 Change Mode Link in the Component Palette



3. If necessary, you can move the link above the Status Indicator by dragging it in the Structure window (Figure 3–38).

Figure 3–38 Change Mode Link in the Structure Window



4. When you add security to your application, as you will in the subsequent Chapter 4, "Adding Security to Your Application,"you can determine which users can edit this page, thus disabling the Change Mode Link you just added. You can adjust the attributes of this link to hide Change Mode Link from users who do not have privileges to edit the page.

   To hide the Change Mode Link from users without Edit privileges, select the link in the Structure Window, then find the **Rendered** property in the Property Inspector under the Common category, as shown in Figure 3–39.

Figure 3–39 Rendered Property in the Change Mode Link Property Inspector

5. You can use the Expression Builder to specify how you want components to display in your application. Alternatively, if you are more comfortable coding, you can go to the Source view of the page, then manually enter the code.

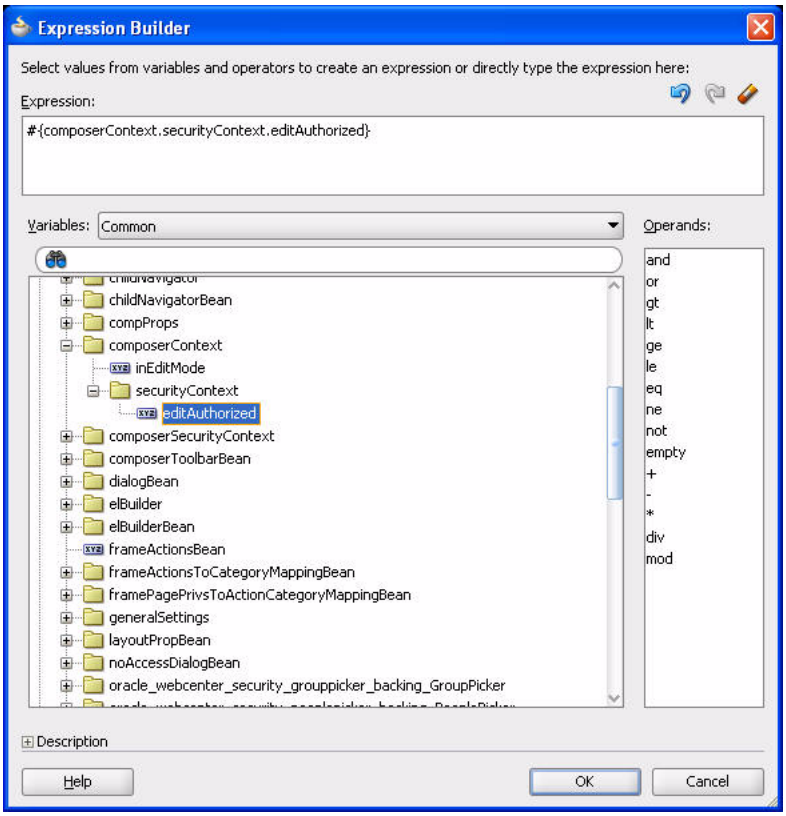
   To the right of the Rendered property, click the down arrow, then choose **Expression Builder** from the context menu.

6. In the Expression Builder, navigate to **JSF Managed Beans**, **composerContext**, **inEditMode**, **securityContext**, then click **editAuthorized**. You'll notice that doing so updates the Expression field at the top of the dialog (Figure 3–40). Example 3–1 shows the code you just created, which you can alternatively copy and paste into the Expression field.

Example 3–1

```
#{composerContext.securityContext.editAuthorized}
```

Figure 3–40 Expression Builder with the Values Filled In



7. Click **OK** to accept the expression for the Rendered property. Until you implement security in your application, the value of this property does not affect the display of the Change Mode Link.
8. Add a Spacer component between the Change Mode Link and Status Indicator components. From the Component Palette, under ADF Faces, expand **Layout**.
9. Drag and drop a **Spacer** component onto the Structure window between the Change Mode Link and Status Indicator, as shown in Figure 3–41.

Figure 3–41 Adding a Spacer Component

10. 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, choose **Oracle Composer** from the list.
11. Under Common Components, drag and drop **Page Customizable** (Figure 3–42) onto the `second` facet in the Structure window (Figure 3–43).

Figure 3–42 Page Customizable in the Component Palette

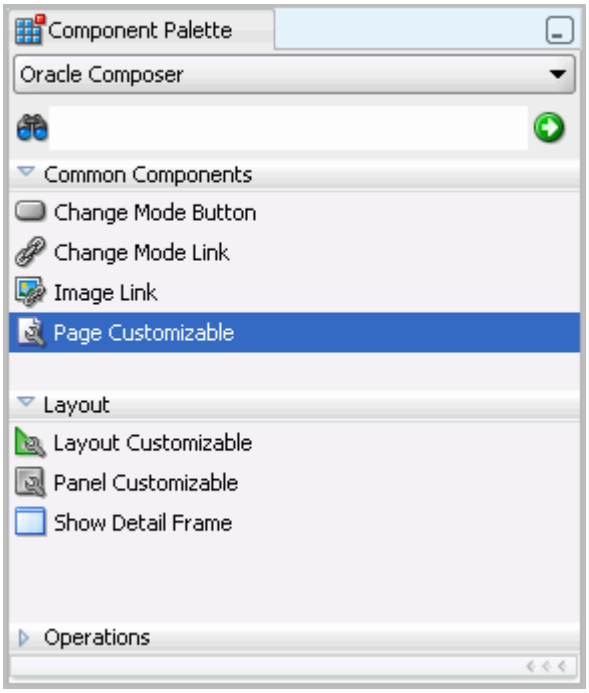
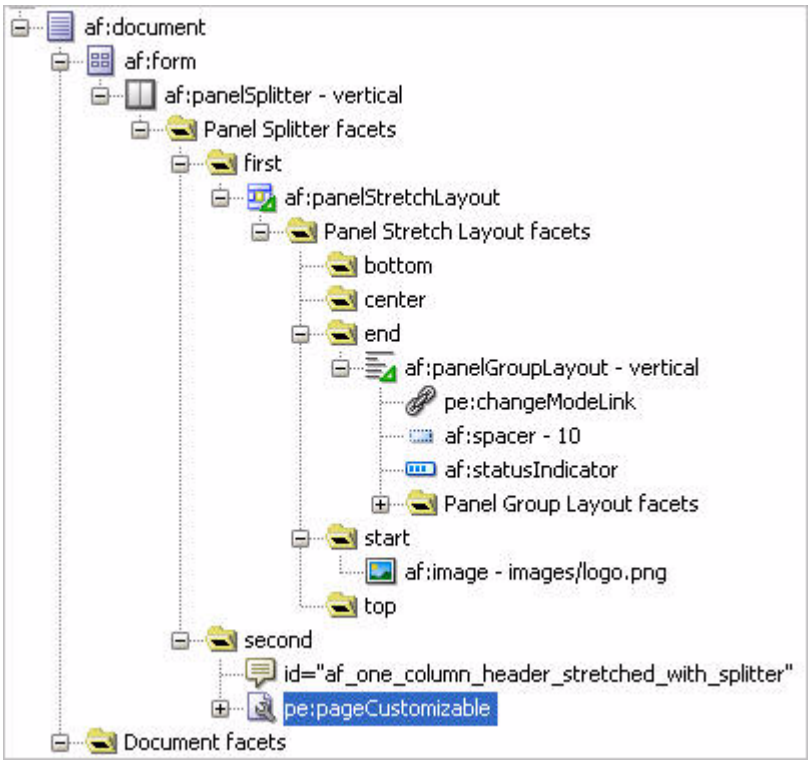
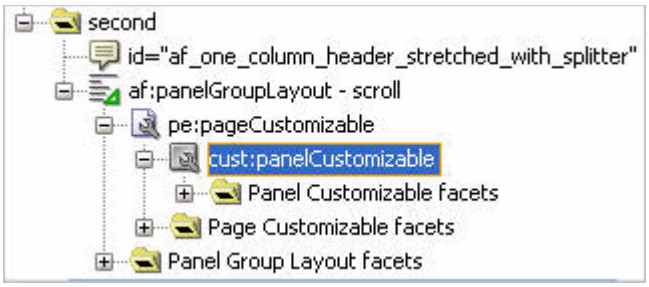


Figure 3–43 Page Customizable in the Second Facet



12. In the Structure window, expand the Page Customizable, then delete the **Panel Customizable** since we do not need it (Figure 3–44). You can delete it by selecting it, then pressing your Delete key.

Figure 3–44 Deleting the Panel Customizable



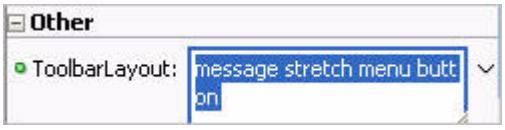
13. Expand the Page Customizable facets node to view the list of components you added. One of these components is a status indicator. This status indicator only displays once you are *in* Composer at runtime. As you will see in the next step, "Step 6: Customize the Page at Runtime Using Oracle Composer", you will see two status indicators: one that you added earlier in this chapter, and another that displays only once you enter Edit mode at runtime in your application.

    To remove this new status indicator, select the **Page Customizable** in the Structure Window.

14. In the Property Inspector for the Page Customizable, expand the **Other** node, then set the **ToolbarLayout** property to the following:

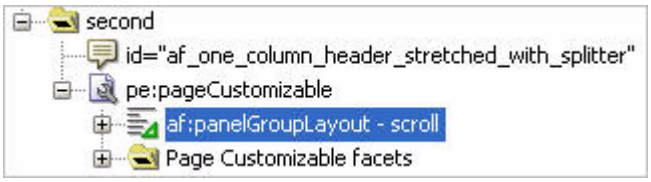
    ```
    message stretch menu button
    ```

Figure 3–45 Removing the Second Status Indicator from the Page Customizable



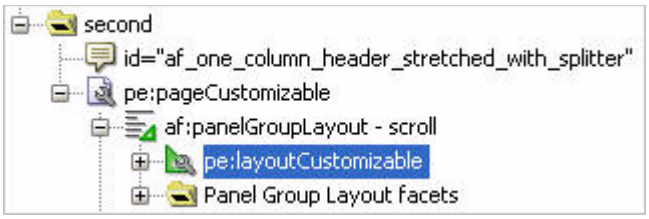
15. Let's continue to add layout components to the page. In the Component Palette, from the ADF Faces list, drag and drop a **Panel Group Layout** component onto the Page Customizable component, then set the **Layout** property to **scroll**. Doing so enables a scrollbar in this area of the layout.

Figure 3–46 Panel Group Layout



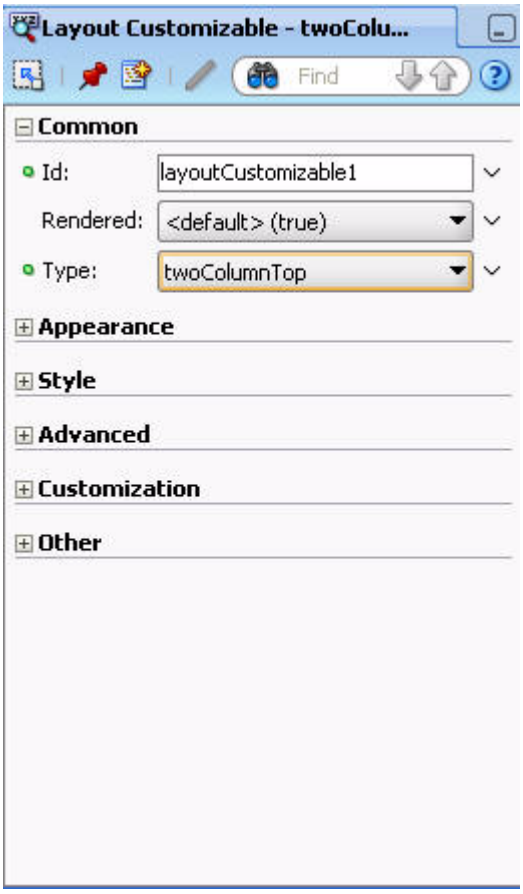
16. Now, we will add a Layout Customizable to our page to create a layout for this region. In the subsequent chapters, we will add services and portlets to this area. From the Component Palette, under Oracle Composer, drag and drop a **Layout Customizable** onto the **Panel Group Layout** in the Structure window (Figure 3–47).

Figure 3–47 Layout Customizable in the Page Customizable



17. While the Layout Customizable is selected, in the Property Inspector, under Common, set the **Type** property to **twoColumnTop** (Figure 3–48).

Figure 3–48 Setting the Layout Customizable Properties



18. Now that you have added a few Oracle Composer components to our page, 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 3–49).

Figure 3–49 MyPage at Runtime with the Change Mode (Edit) Link

For more information on adding Oracle Composer to your application, see the chapters in Part II, "Using, Extending, and Customizing Your Application with Oracle Composer" in *Oracle Fusion Middleware Developer's Guide for Oracle WebCenter*.

We will use Oracle Composer in the application 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 will test a few features that Oracle Composer offers. Later in the Tutorial, after you implement security and add a few features to your application, you will test the application at runtime again for a more in-depth look at personalizing your application.

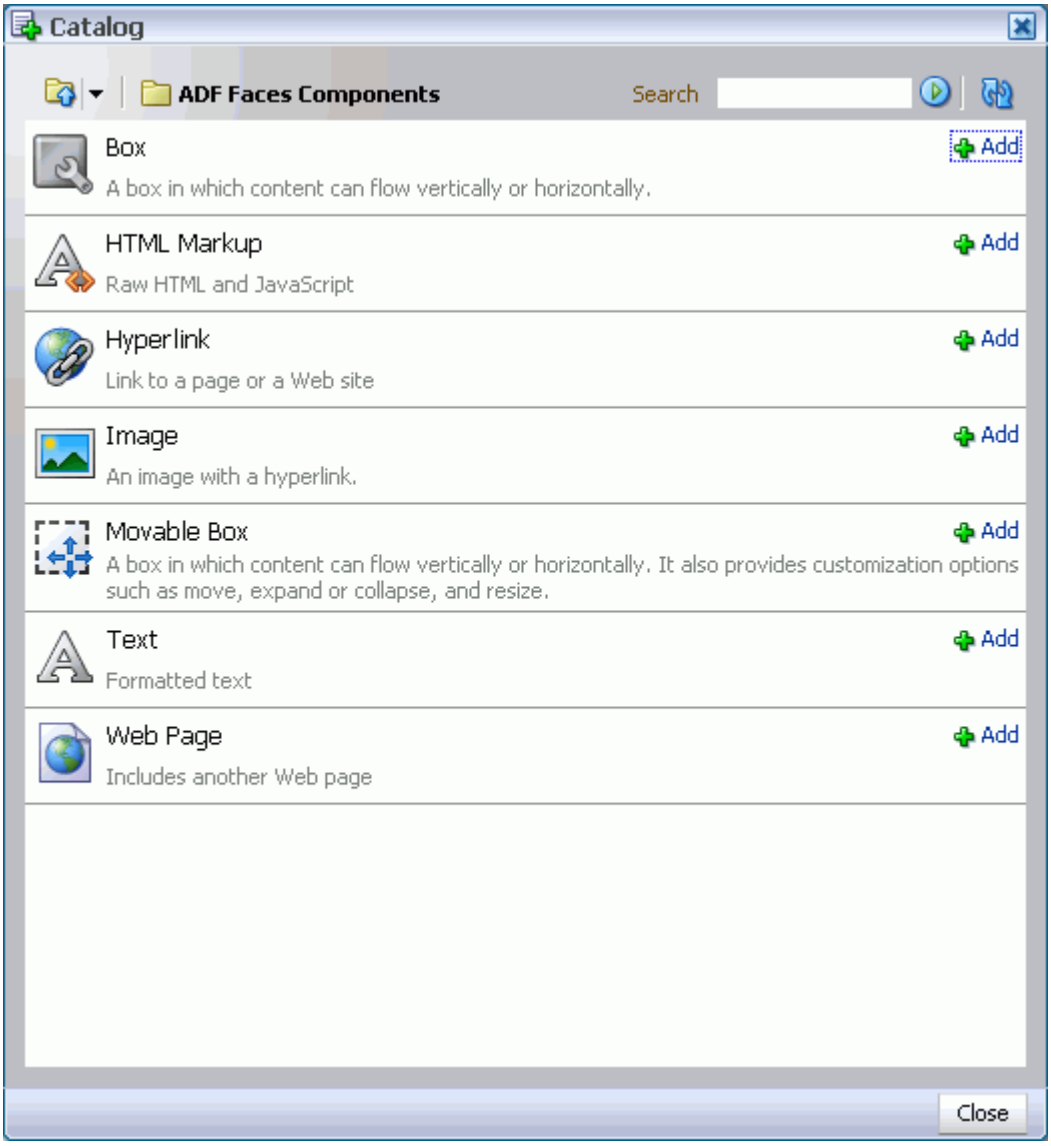
1. While MyPage displays in your browser, you'll notice an Edit link above the Status Indicator. Click **Edit** to see how the page looks in Edit mode (Figure 3–50).

Figure 3–50 Edit Mode of MyPage

2. Take a quick look at what you can do in Edit mode.

In the top left horizontal region, click **Add Content** to view the runtime catalog. Here, you can click **ADF Faces Components** to see the different ADF Faces Components you can add at runtime (Figure 3–51).

Figure 3–51 Adding Content at Runtime

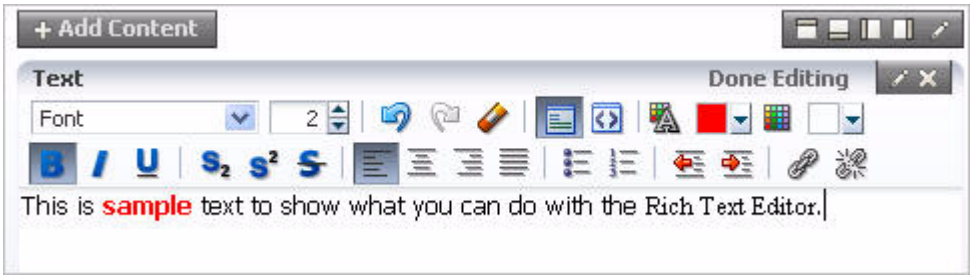


3. Add a Text object to see how it looks. In the Catalog, next to Text, click **Add**, then click **Close**. A text box now displays in the region, as shown in Figure 3–52.

Figure 3–52 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 3–53.

Figure 3–53 Entering and Modifying Text in the Rich Text Editor at Runtime



6. When you are finished, click **Done Editing**.
7. You can change the overall layout of your page by clicking **Change Layout**, then choosing a layout option (Figure 3–54).

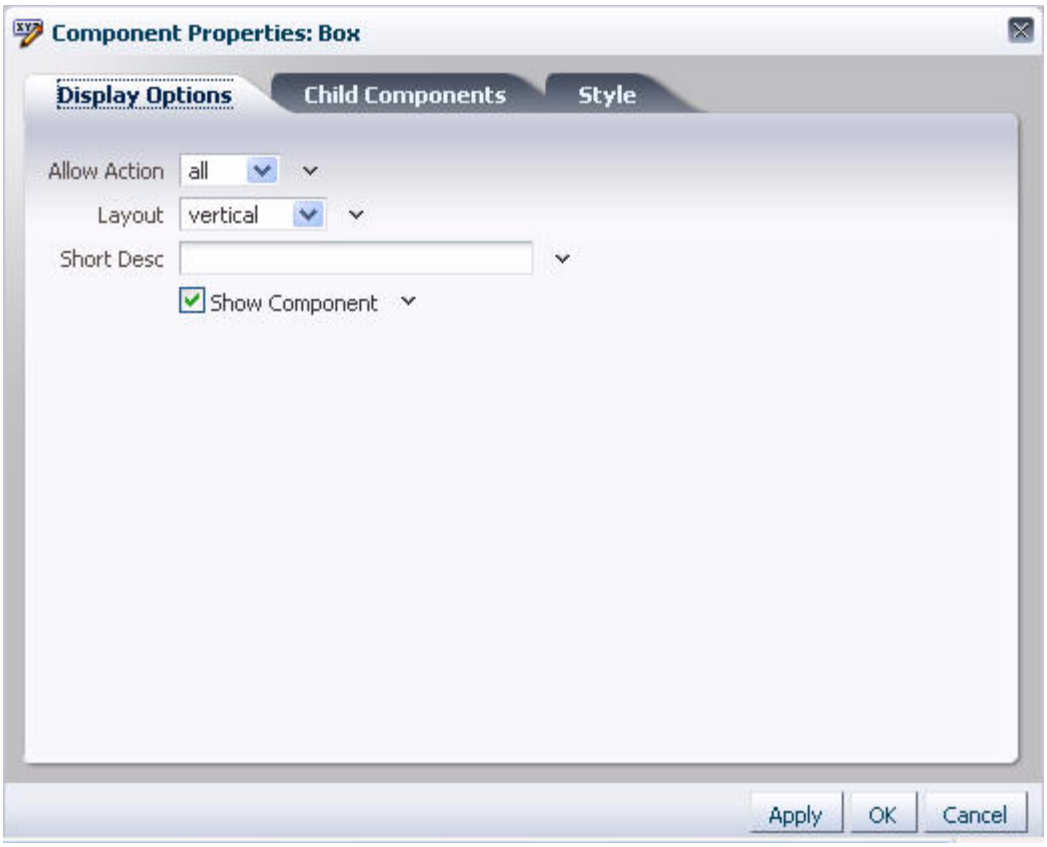
Figure 3–54 Changing the Layout at Runtime

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 3–55.

Figure 3–55 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 3–56).

Figure 3–56 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 panel.

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.

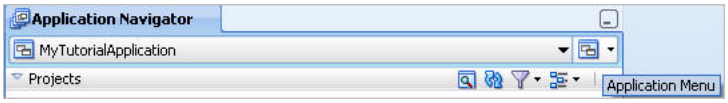
11. Return to Oracle JDeveloper.

    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.

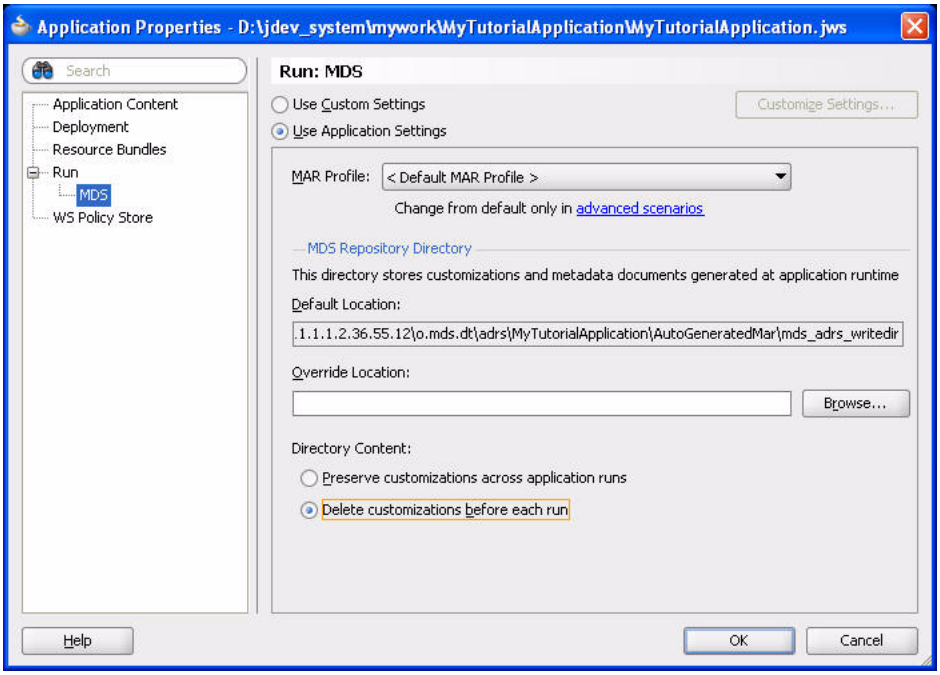
    In the Application Navigator, next to MyTutorialApplication (in the list), click the **Application Menu** icon.

Figure 3–57 Application Menu



13. From the context menu, choose **Application Properties...**.
14. In the Application Properties dialog, expand the **Run** node in the left pane, then select **MDS**.
15. In the right pane, under Run: MDS, select **Delete customizations before each run** (Figure 3–58). Doing so clears the changes you made when you entered Edit mode at runtime the next time you run the application. Other changes you make at runtime (for example, in the services you add in Chapter 5, "Adding Oracle WebCenter Services to Your Application") are not stored in the MDS, and thus this option does not clear those change every time you run the application.

Figure 3–58 Application Properties Dialog



16. Click **OK**.

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

Now that we've created a custom WebCenter application added the resource files to your application, built a customizable JSF page, and taken a quick tour of Oracle Composer at runtime, you can add security to your application in Chapter 4, "Adding Security to Your Application."

4

Adding Security to Your Application

In this lesson, you will add basic security to your custom WebCenter application, then create three different sample users who can log into the application. You will also add elements to the header of MyPage, so that users can click a login link to access your application, as well as a login page that displays the username and password fields.

By enabling security in the development environment of your application, you can test security-based features, such as logging in as a particular user to check email, or logging in as the administrator to make a change to the overall application. This chapter shows you how to set up security; in the next chapter, you will see how you can leverage WebCenter services that rely on security.

At the end of this lesson, the page you created in Chapter 3, "Creating a WebCenter Application with a Customizable Page" will look like Figure 4–1.

Figure 4–1 MyPage.jspx in the Browser

Introduction

This lesson contains the following steps:

- Step 1: Add ADF Security to Your Application
- Step 2: Create Users and Roles for the Application
- Step 3: Add ADF Security Policies to Your Application
- Step 4: Add a Login/Logout Link to Your Application and Update the Login Page

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

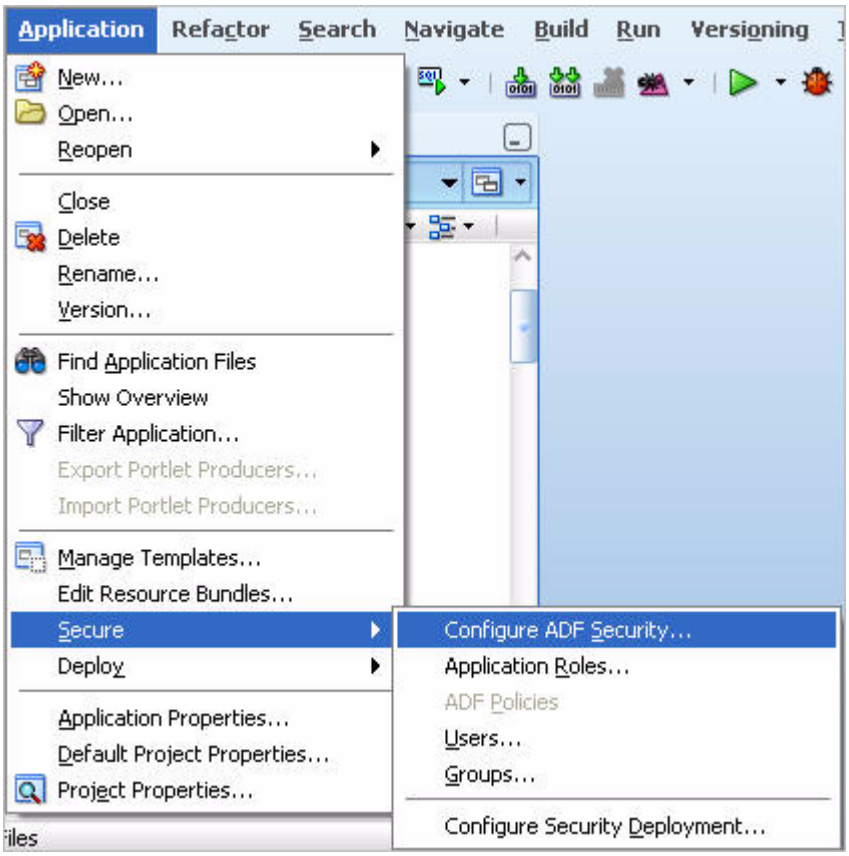
Step 1: Add ADF Security to Your Application

Oracle JDeveloper includes an ADF Security wizard that enables you to add basic security to your application. In this step, we use this wizard to add security to our existing application.

To add ADF security:

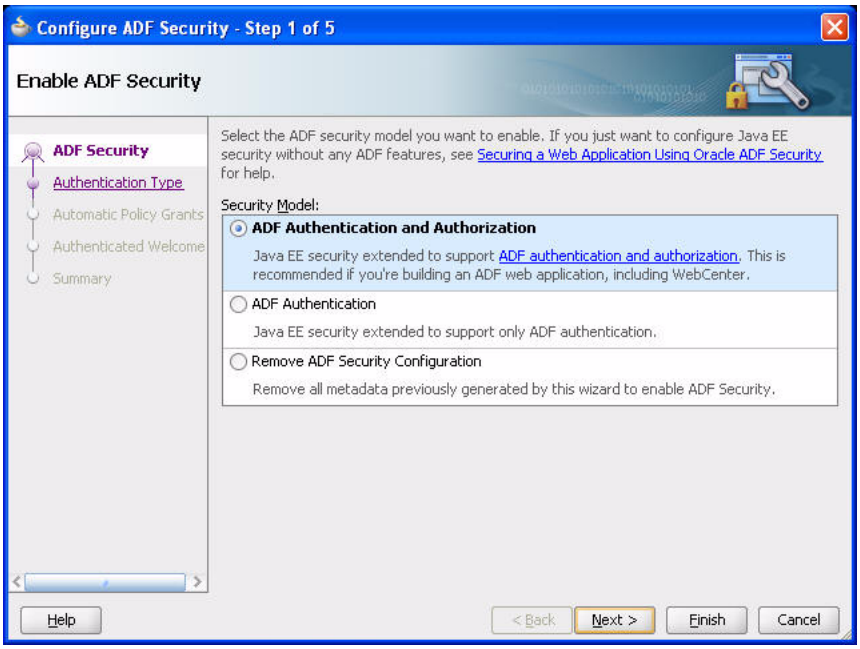
1. In JDeveloper, while the application is open, choose **Application** from the main menu, then select **Secure**, and **Configure ADF Security** to display the Configure ADF Security wizard.

Figure 4–2 Configure ADF Security Menu Option



2. On the Enable ADF Security page, ensure **ADF Authentication and Authorization** is selected. Choose this option when securing any ADF web application, such as a custom WebCenter application (Figure 4–3).

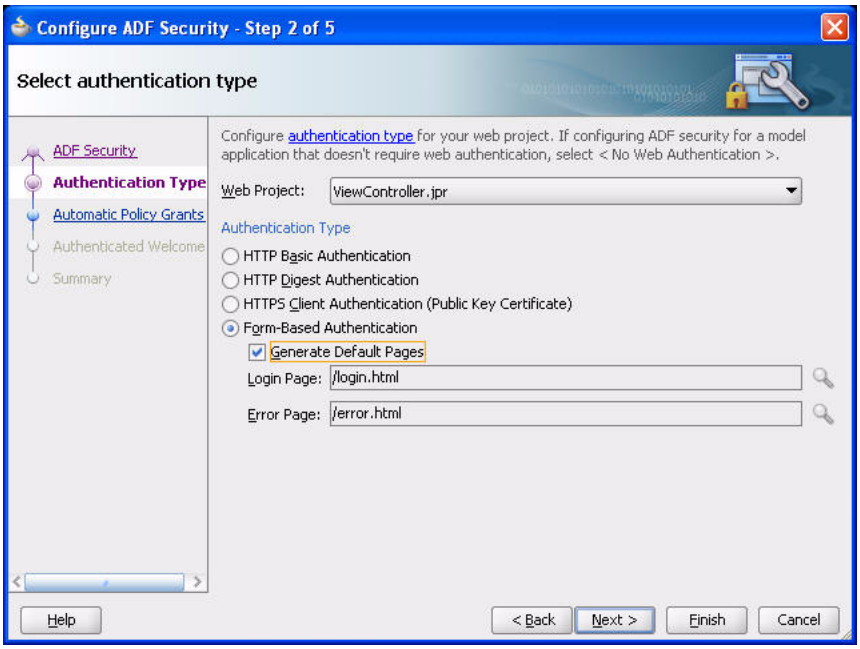
Figure 4–3 Configure ADF Security - Step 1 of 5



3. Click **Next**.
4. On the Select authentication type page, ensure **Form-Based Authentication** is selected. Choosing this option generates a login page where users can enter their username and password for the application.
5. Select **Generate Default Pages**, and leave the default page names: `/login.html` and `/error.html` (Figure 4–4).

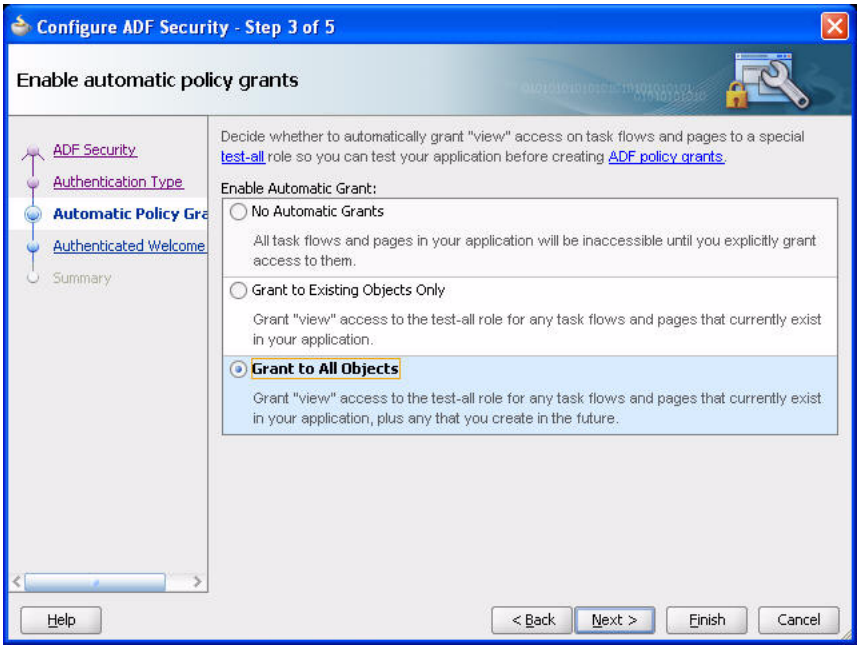
   Click **Next**.

Figure 4–4 Configure ADF Security - Step 2 of 5



6. Click **Next**.
7. On the Enable automatic policy grants page, ensure **Grant to All Objects** is selected. Doing so enables the `test-all` role in your application View access to any pages you create in the application (Figure 4–5).

Figure 4–5 Configure ADF Security - Step 3 of 5



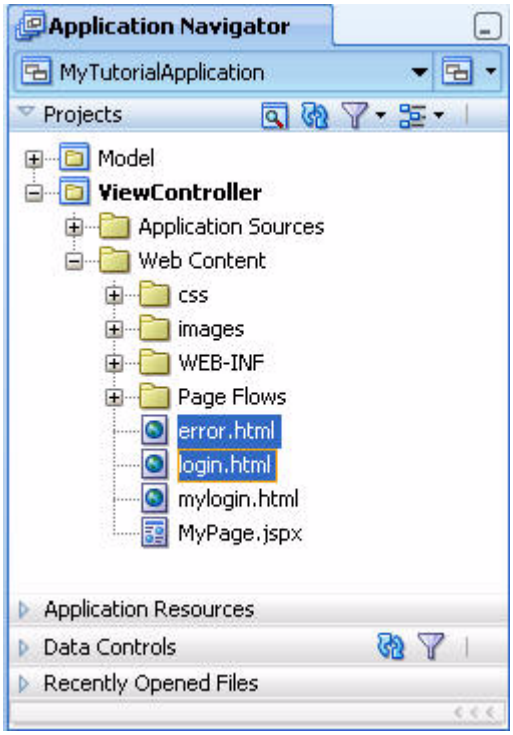
8. Click **Next**.

9. On the Specify authenticated welcome page, click **Next**. You can learn more about this option in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

10. On the Summary page, notice the files that the wizard will create or modify based on your selections, then click **Finish**. If a dialog displays, click **OK**.

    In the Application Navigator notice that two new pages display in the ViewController project, under WebContent: **error.html** and **login.html**.

Figure 4–6 Security Files in the Application Navigator



For more information about testing security during development and WebCenter application security, see Chapter 3, "Securing Your WebCenter Application" in *Oracle Fusion Middleware Developer's Guide for Oracle WebCenter*.

Now that you have set up basic ADF security for the application, you can create the users and roles.

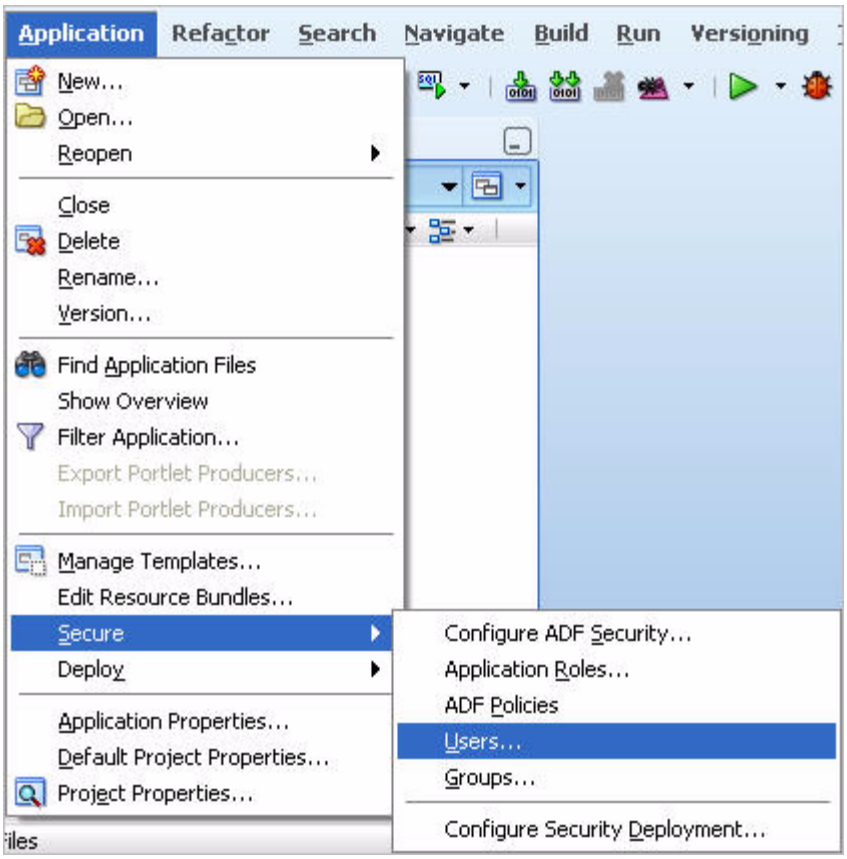
Step 2: Create Users and Roles for the Application

Now that we have added basic ADF security to our application, we can create sample users to test the authentication. In this section, you will create three users: a user with administrative privileges who can make changes to the entire application, a user who can only make modifications to his own view of the application, and a user who can make some modifications to the application in addition to his own view. This step introduces you to the `jazn-data.xml` file, which contains the security information for your application.

To create users for the application:

1. From the Application menu, choose **Secure**, then **Users** to display the `jazn-data.xml` file.

Figure 4–7 Creating Users

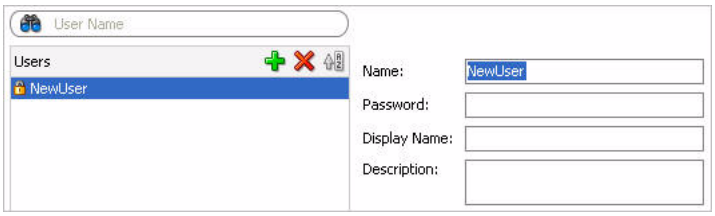


2. In the `jazn-data.xml` file, ensure that the **Users** tab is selected.
3. In the Users list, click the **New User** icon (Figure 4–8) to add a user temporarily named "NewUser" to the Users list.

Figure 4–8 New Users Icon

4. While NewUser is selected, notice that the right pane updates so that you can modify the properties of the user (Figure 4–9).

Figure 4–9 NewUser Properties

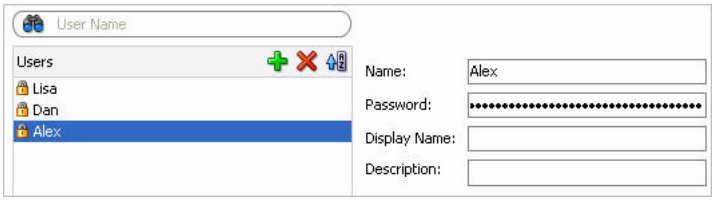


In the Name field, enter `Lisa`.

5. In the password field, enter `welcome1` then click on the Display Name field to make sure the password is accepted. Leave the rest of the fields blank for now.

6. Create two more users by following steps 3 through 5:
   - User: `Dan`, Password: `welcome1`
   - User: `Alex`, Password: `welcome1`

Figure 4–10 New Users in the jazn-data.xml File



7. Now that we have created our sample users, we can create *roles* for the application, then assign the users to the roles. In the left pane, click the **Application Roles** tab (Figure 4–11). Notice that a role is already listed, called `test-all`. This role is automatically generated by the Configure ADF Security wizard. You can use this role for testing purposes, but you will create your own roles for this application.

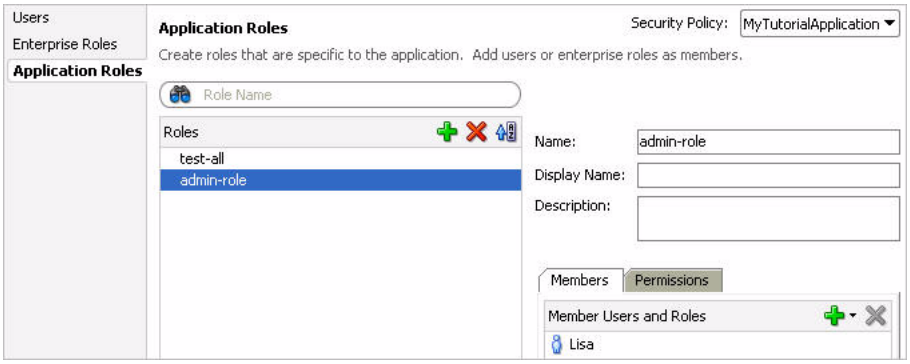
Figure 4–11 Application Roles Tab

8. In the Roles list, click the **New Application Role** icon.
9. While NewApplicationRole is selected, in the Name field to the right, enter `admin-role`.
10. In the Display Name field, enter `Administrators`.
11. Click the **Members** tab, then click the **Add User or Role** icon, then select **Add User** from the menu.
12. In the Select Users dialog, click **Lisa**, then click **OK**.

Figure 4–12 Select Users Dialog

The new role displays with the user `Lisa` listed.

Figure 4–13 Admin-Role



13. Create another role called `user-role` and set the Display Name to `Users`.
14. Add the users **Dan** and **Alex** to this role.
15. Save all your files.

For more information about users and roles, see *Oracle Fusion Middleware Developer's Guide for Oracle WebCenter* and Chapter 3, "Securing Your WebCenter Application" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

Step 3: Add ADF Security Policies to Your Application

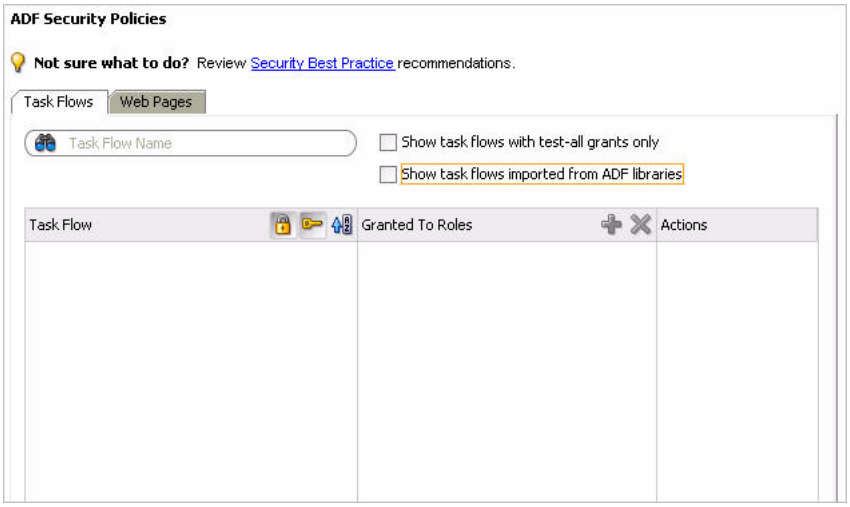
Once you set up ADF security for your application using the Configure ADF Security wizard and set up your users, you must create the ADF security policies for your application. When you add ADF security policies to your application, you apply them to the page that requires authentication. The ADF security policies indicate the permissions for the application roles you set in the previous step. That is, the security policies define the actions that different users can perform on various objects in the application, such as pages and task flows. For example, you can set MyPage to be viewable by any user who is a member of the `user-role` and customizable by any user who is a member of the `admin-users` role.

To add ADF security policies to your application:

1. Before we can add ADF security policies to the application, we must first create a page definition for MyPage. Then, we will add the application roles and set their permissions on that page.

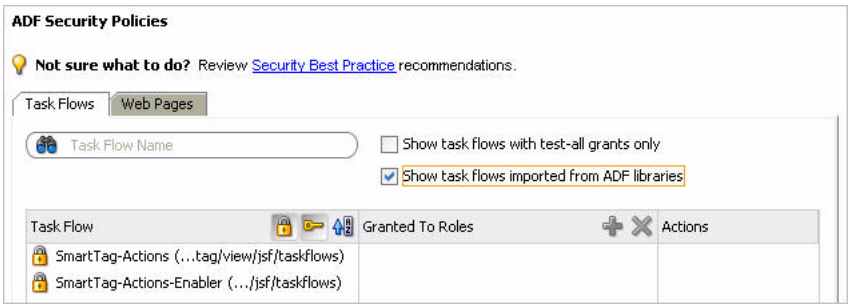
   While MyPage is displaying in the Design view (you may need to click its tab to bring it into focus), right-click the page, then choose **Go to Page Definition**.
2. If the Confirm Create New Page Definition dialog displays, click **Yes**. The page definition file, called `MyPagePageDef.xml` displays. You can close this tab for now.
3. Click the **MyPage.jspx** tab to bring it into focus.
4. From the Application menu, choose **Secure**, then **ADF Policies** to display the ADF Security Policies section of the `jazn-data.xml` file, as shown in Figure 4–14.

Figure 4–14 ADF Security Policies



5. Before adding the ADF security policies to our application, take a quick look at the Task Flows you currently have in the application. Select the **Show task flows imported from ADF libraries** checkbox.

Figure 4–15 ADF Security Policies for Task Flows

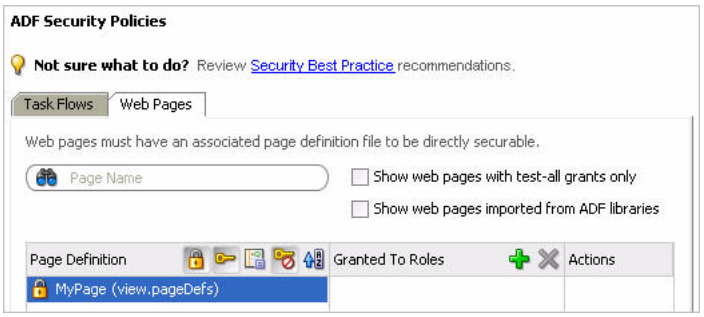


These task flows are generated by default for your application. Once you start adding task flows in Chapter 5, "Adding Oracle WebCenter Services to Your Application," this list will automatically be updated with the new task flows.

6. Next, add the necessary ADF Security Policies to the application. At the top of the section, click the **Web Pages** tab.

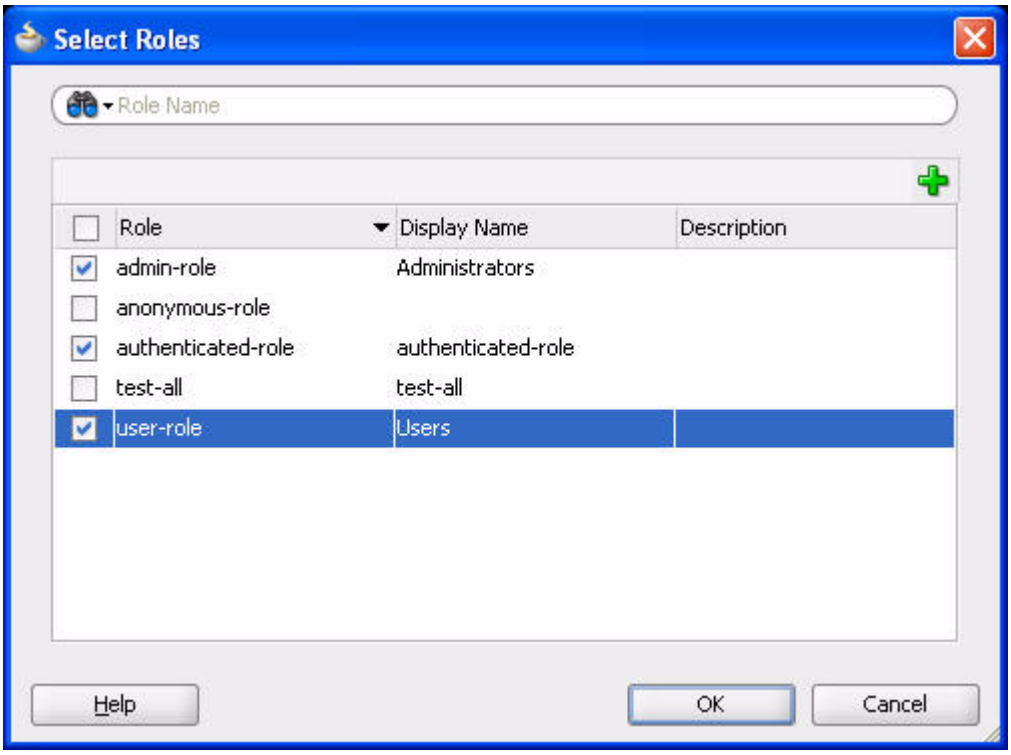
7. In the Page Definition list, click **MyPage** (Figure 4–16).

Figure 4–16 MyPage on the ADF Security Policies Section



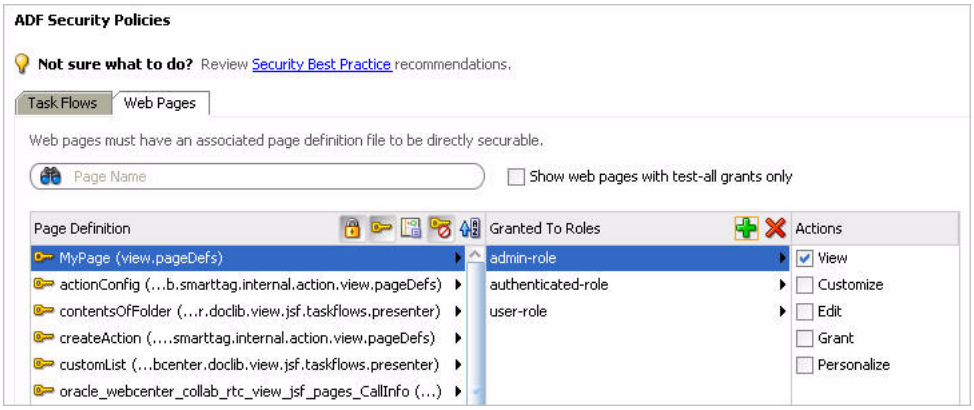
8. Next to the Granted To Roles column, click the **Add Application Role** icon.
9. In the Select Roles dialog, select **admin-role**, **authenticated-role**, and **user-role**, then click **OK** (Figure 4–17).

Figure 4–17 Select Roles Dialog



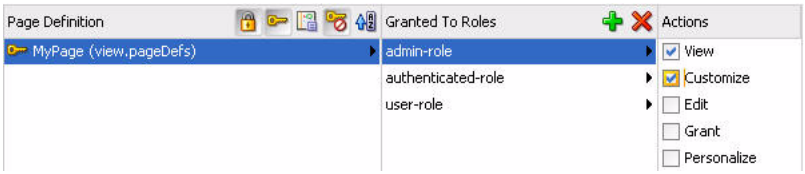
The three roles now display on the Web Pages tab of the ADF Security Policies page (Figure 4–18).

Figure 4–18 MyPage ADF Security Policies



10. Next, assign the permissions each role has for MyPage. While `admin-role` is selected, under **Actions**, select **View** and **Customize**, as shown in Figure 4–19.

Figure 4–19 Assigning the View and Customize Actions to the admin-role



Doing so allows any user with the admin-role (in our example, Lisa) to view the page and customize it. Any *customizations* that Lisa makes will proliferate to the views of all users. Customizations are different from *personalizations*; the latter can only be viewed by the currently authenticated user.

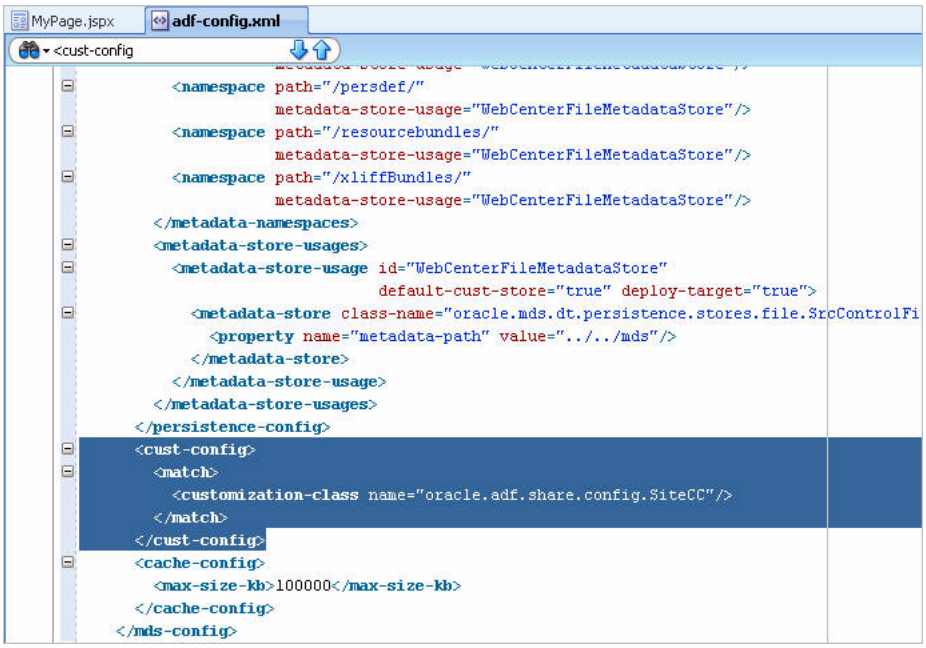
11. Select the **authenticated-role** and ensure the role has **View** permissions for MyPage.
12. Select the **user-role** and select the **View** and **Personalize** actions.
13. Finally, we must make one change to the `adf-config.xml` file to set any customizations that the user makes to the user level and not the site or application level.
14. In the Application Navigator, in the Application Resources panel, expand **Descriptors**, then **ADF META-INF**.

Figure 4–20 adf-config.xml File in the Application Resources Panel

15. Open the `adf-config.xml` file, and switch to the Source view by clicking the **Source** tab at the bottom of the page.
16. Locate the following code snippet (as shown in Figure 4–21), which only displays if you have added customizable components from Oracle Composer to your page, as you did in Chapter 3, "Creating a WebCenter Application with a Customizable Page":

```
<cust-config>
  <match>
    <customization-class name="oracle.adf.share.config.SiteCC"/>
  </match>
</cust-config>
```

Figure 4–21 Section to Overwrite in the adf-config.xml File



17. Replace the code snippet with the following code snippet:

```
<cust-config>
  <match>
    <customization-class name="oracle.adf.share.config.UserCC"/>
  </match>
</cust-config>
```

18. Save all your files.

For more information about ADF Security Policies, see *Oracle Fusion Middleware Developer's Guide for Oracle WebCenter*.

Now that you have set up security for your application, you can enable users to log in and out of the application by adding a link to MyPage.

Step 4: Add a Login/Logout Link to Your Application and Update the Login Page

To enable your users to log in and out of the application, in this step, you will add a Login/Logout link to the upper right corner of the header that toggles depending on whether the user is authenticated.

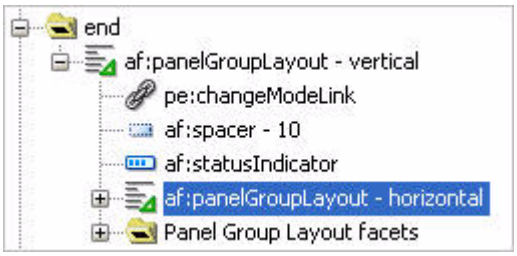
Figure 4–22 Logout Link in the Header

To add a login/logout link:

1. In Oracle JDeveloper, bring `MyPage.jspx` into focus.
2. In the Structure window for `MyPage.jspx`, navigate to the Panel Splitter, then open the `first` facet. Remember that 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).
3. Under the `first` facet, expand **af:panelStretchLayout**, open the **Panel Stretch Layout facets** folder, then open the **end** folder.
4. First, organize the components you already have into a layout so that we can add the login/logout link. Place a Panel Group Layout onto the existing vertical Panel Group Layout that contains the Change Mode Link you added in "Step 5: Add Oracle Composer to the Page to Enable Customization" in Chapter 3, "Creating a WebCenter Application with a Customizable Page."

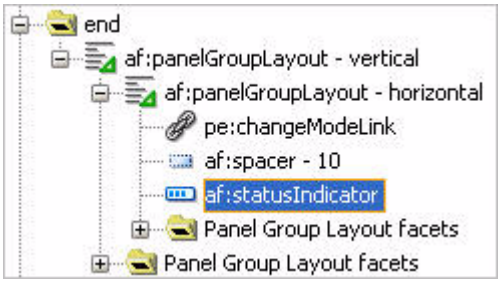
   To do so, in the Component Palette, select **ADF Faces** from the list.
5. Under Layout, drag and drop **Panel Group Layout** onto the `af:panelGroupLayout-vertical`, and set the Layout property to **horizontal**.

Figure 4–23 New Panel Group Layout with Horizontal Layout



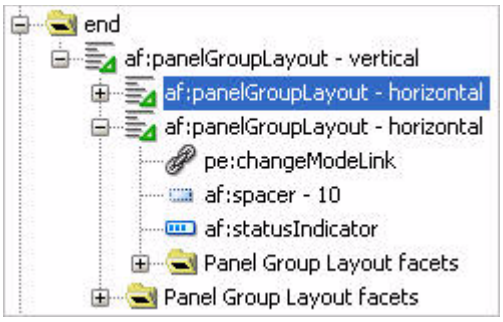
6. Drag and drop the **Change Mode Link**, **Spacer**, and **Status Indicator** into the new horizontal Panel Group Layout.

Figure 4–24 Horizontal Panel Group Layout with the Components



7. Drag and drop another **Panel Group Layout** onto the vertical Panel Group Layout and set its **Layout** property to **horizontal**.
8. Move the new **Panel Group Layout** above the existing horizontal Panel Group Layout that contains the Change Mode Link, Spacer, and Status Indicator (Figure 4–25).

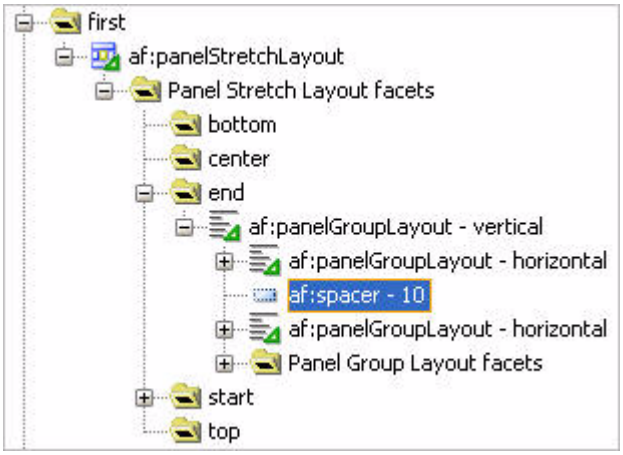
Figure 4–25 Second Horizontal Panel Group Layout



9. To lay out the components in the header, add a Spacer component between the two Panel Group Layout components. From the Component Palette, under ADF Faces, expand **Layout**.

10. Drag and drop a **Spacer** component onto the Structure window between the two horizontal Panel Group Layout components (Figure 4–26).

Figure 4–26 Spacer Component in the Structure Window



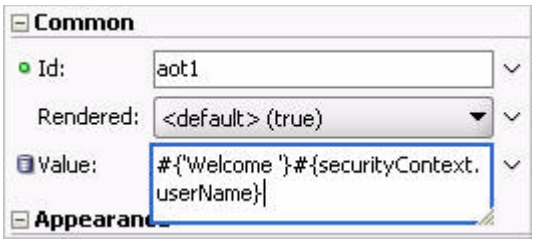
11. Add a Welcome message that displays the name of the currently authenticated user.

    Drag and drop an **Output Text (Active)** component from the Component Palette (under ADF Faces, Common Components) onto the first `af:panelGroupLayout-horizontal`.

12. In the Property Inspector for the Output Text, in the Value field, enter the following code snippet, as shown in Figure 4–27:

    ```
    #{'Welcome '}#{securityContext.userName}
    ```

Figure 4–27 Output Text Value



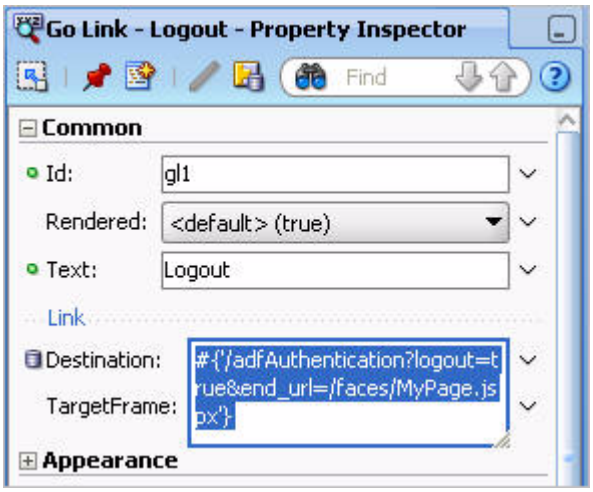
13. Add a spacer after the Welcome text. From the Component Palette, under ADF Faces, drag and drop the **Spacer** component onto the Structure window.
14. Next, add a Logout link. From the Component Palette, under ADF Faces, drag and drop a **Go Link** component just below the Spacer component in the Structure window, and set the **Text** property to `Logout`.

Figure 4–28 Go Link

15. Set the **Destination** property to the following, as shown in Figure 4–28:

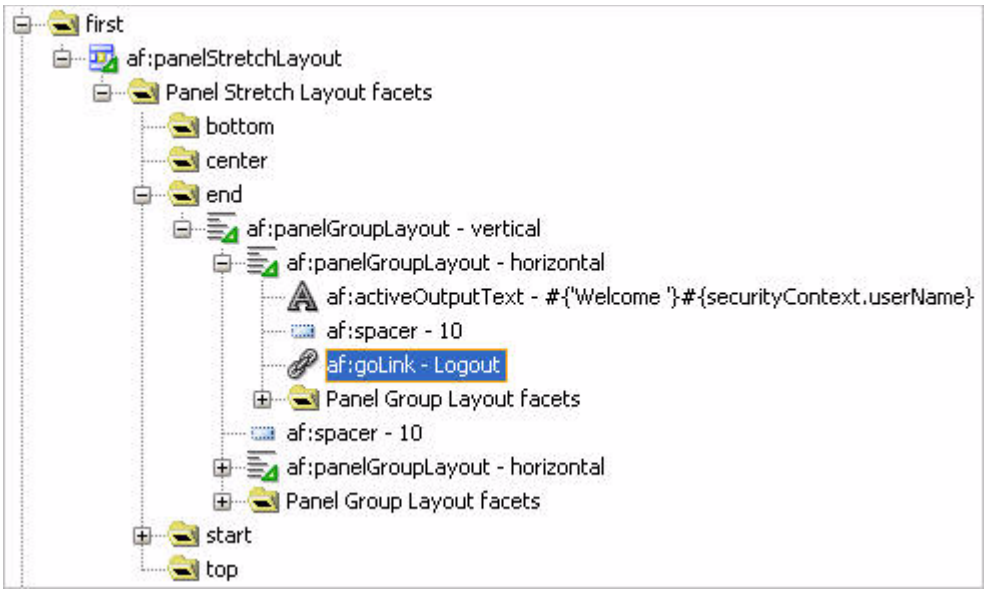
```
#{'/adfAuthentication?logout=true&end_url=/faces/MyPage.jspx'}
```

Figure 4–29 Destination Property for the Go Link Component



16. Save the page. The structure for the `first` facet should look like:

Figure 4–30 Structure WIndow with the Output Text and Logout Link



17. When you implemented security using the ADF Security Wizard in "Step 1: Add ADF Security to Your Application", you generated two pages: `error.html` and `login.html`. The `error.html` file displays a message if a user tried to log into the application, but is unsuccessful. The `login.html` file displays a username and password field where the user can authenticate with the application.

    In JDeveloper, in the ViewController project, open the **Web Content** folder. You should see the `login.html` page display just above `MyPage.jspx`.

18. Open the **login.html** file (Figure 4–31). This login page should display when you run `MyPage.jspx` to your browser.

Figure 4–31 Default Login.html Page

19. Instead of using this default page, you can use the login page you added to your application resources in Chapter 3, "Creating a WebCenter Application with a Customizable Page."

20. In the Application Navigator, under ViewController, expand the **WEB-INF** folder.

21. Open the `web.xml` file.

22. While viewing the Overview of this file, click the **Security** tab on the left side.

Figure 4–32 Security Tab of the web.xml File

23. Under Login Authentication, ensure **Form-Based Authentication** is selected.
24. Next to the Login Page field, click the **Browse...** icon.
25. Navigate to the **public_html** folder containing `mylogin.html` and select the file.
26. Click **OK**. The Security page updates to use the new `mylogin.html` file.

Figure 4–33 Security Tab of the web.xml File Showing the New Login Page

27. Take a look at security at runtime. Run `MyPage.jspx` to your browser.
28. In your browser, in the Username field, enter `Lisa` with the password `welcome1` (Figure 4–34) and click **Submit**.

Figure 4–34 Logging into Your Application

The page displays in your browser (Figure 4–35):

Figure 4–35 MyPage.jspx in the Browser

Now that you have implemented security on the application, you can add content to the page in Chapter 5, "Adding Oracle WebCenter Services to Your Application."

5

Adding Oracle WebCenter Services to Your Application

Oracle WebCenter Framework includes Oracle WebCenter Services, which enable you to enhance your application with 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 and a document library to your application. With the document library, or the Documents service, you can enable your users to share and manage content like Word documents or PDFs in an easy-to-navigate environment, right in the application. Additionally, you will take advantage of the People Connections service to examine how users can create a social network and keep up to date with each other by viewing recent activities and messages posted by the users.

You will also add the Tags service, which 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. Then, you will add the Links service, which enables you to create relationships between items in your application, as well as relationships between items in your application and other items, like external URLs.

Optionally, if you have access to an email server, you can add the Mail service to the sample application to see how users can email each other within the application at runtime.

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 Services, refer to the *Oracle Fusion Middleware Developer's Guide for Oracle WebCenter*.

Figure 5–1 shows the Search toolbar, document library, and Tag Cloud that you will create in this lesson. Figure 5–2 shows the People Connections service on MyPage in the application.

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

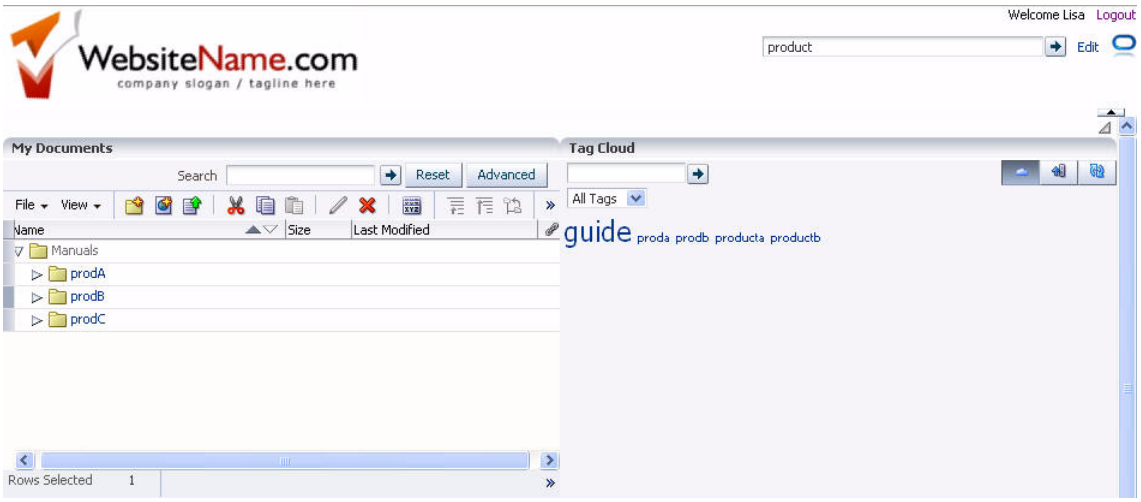
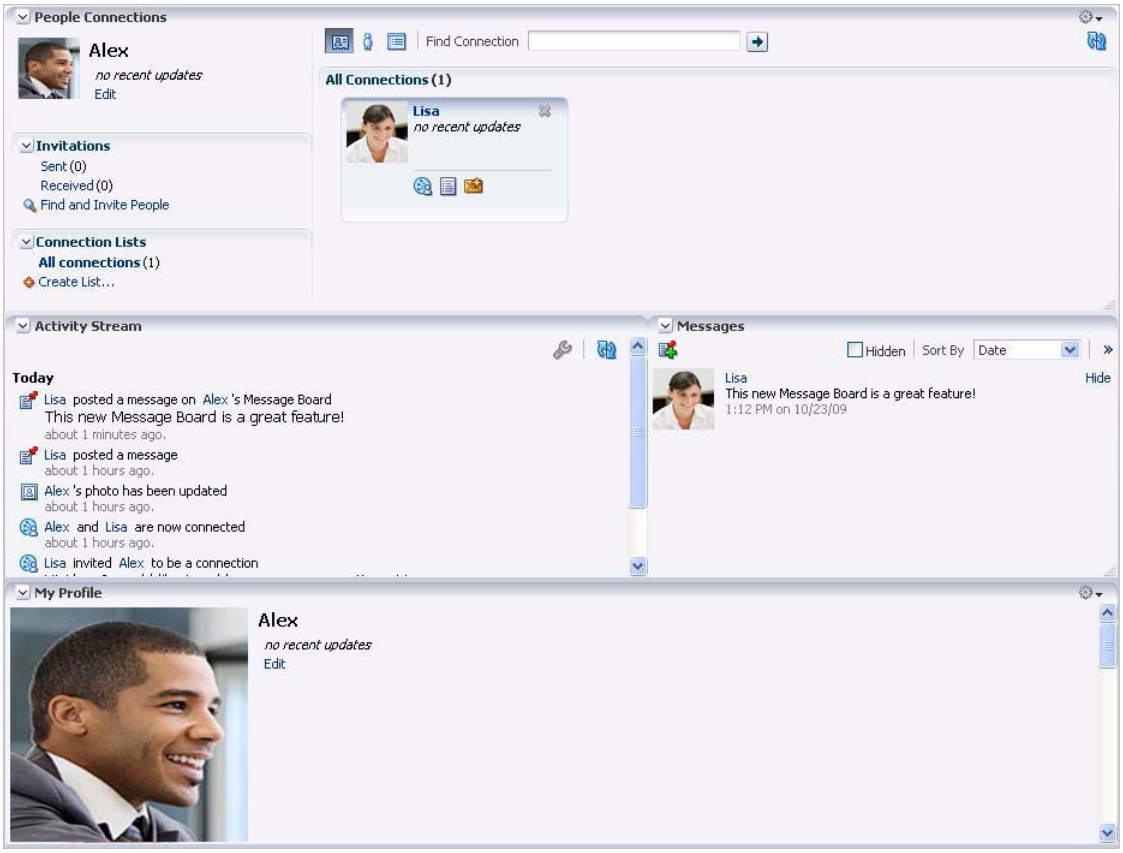


Figure 5–2 People Connections Section of MyPage at the End of this Lesson (Logged in as Alex)



Introduction

This lesson contains the following steps:

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

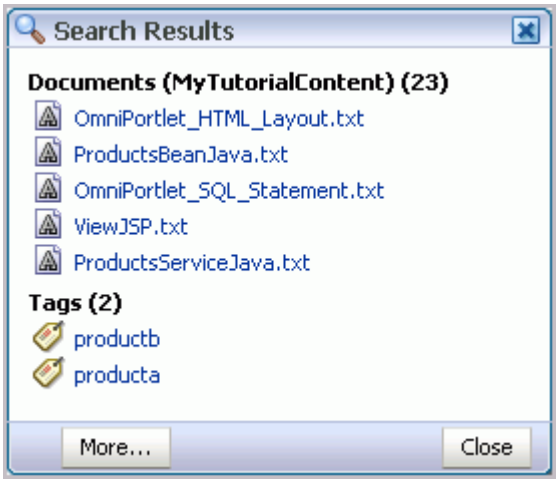
- Step 2: Create a Connection for the Documents Service
- Step 3: Add the Documents - Document Manager Task Flow to Your Application
- Step 4: Browse Documents at Runtime
- Step 5: Create a Database Connection to the WebCenter Schema
- Step 8: Add the People Connections Service to Your Application
- Step 9: Use the People Connections Service in Your Application at Runtime
- Step 6: Add the Tags Service to Your Application
- Step 7: Use, Add, and Search Tags in Your Application at Runtime
- Step 10: Use the Links Service in Your Application at Runtime
- Step 11: Use the Mail Service with Your Application (Optional)

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 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 5–3 shows a sample of the search results you can retrieve in your application.

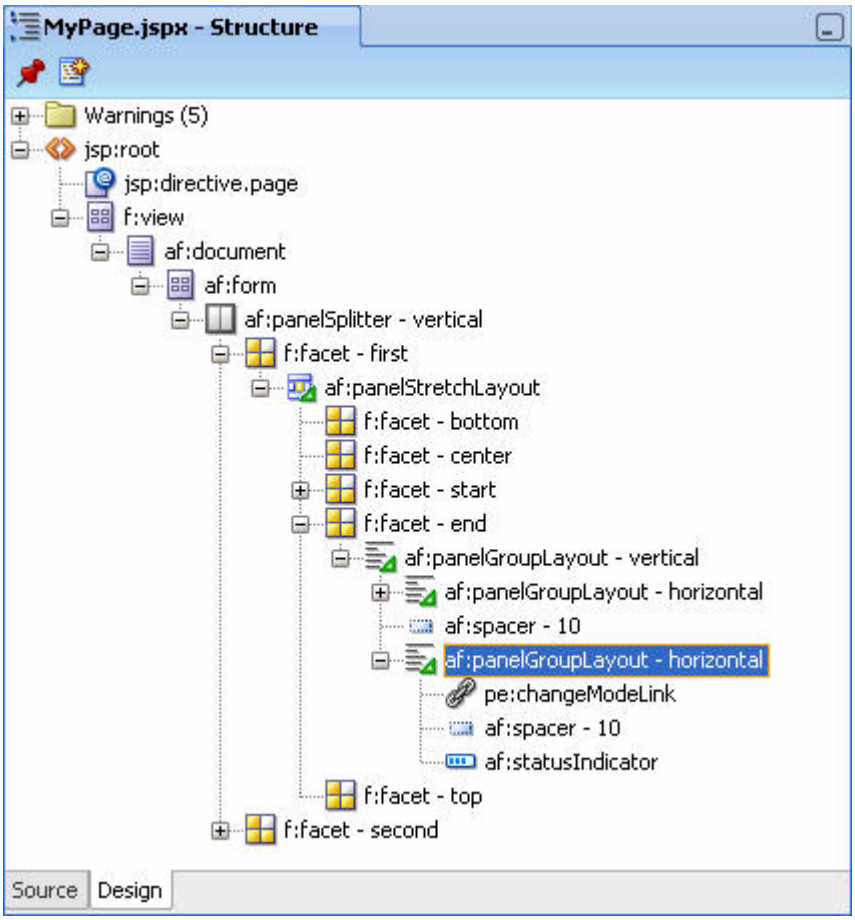
Figure 5–3 Sample Search Results



To add the Search service to your application:

1. In Oracle JDeveloper, ensure **MyTutorialApplication** is open and that your page, `MyPage.jspx`, is open and visible.
2. In the Structure window, locate the second horizontal Panel Group Layout (in the `end` facet) that you added in Chapter 4, "Adding Security to Your Application." Figure 5–4 shows the Panel Group Layout selected. You will add the Search Toolbar task flow here.

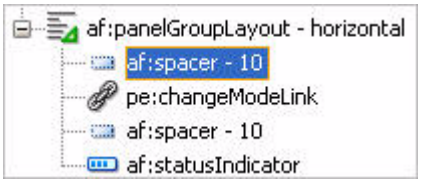
Figure 5–4 Panel Group Layout in the end Facet



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. Before you add the Search Toolbar task flow, add a Spacer component to create the layout. From the Component Palette, under **ADF Faces**, under **Layout**, drag and drop a **Spacer** component onto the Structure window just above the Change Mode Link (Figure 5–5).

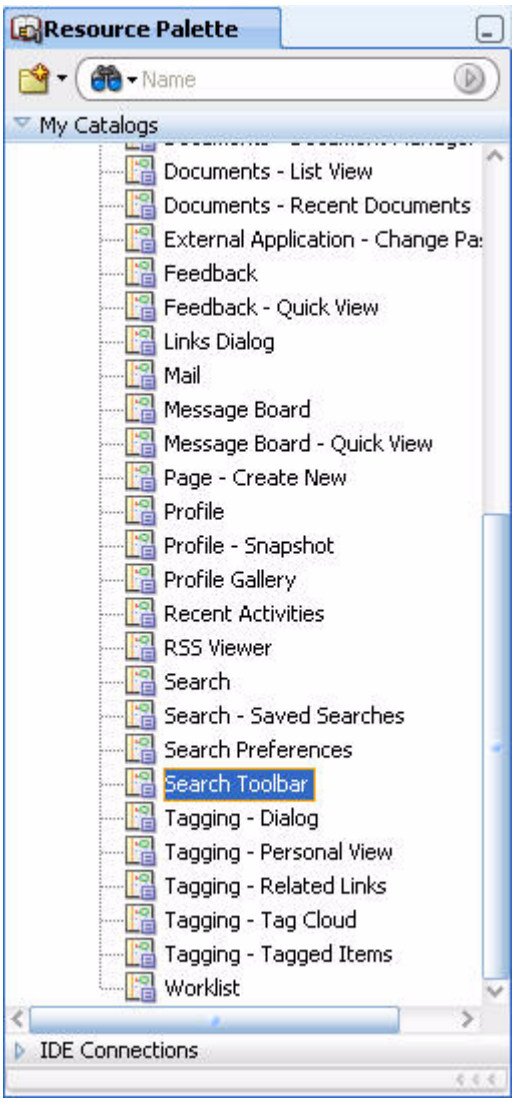
Figure 5–5 Spacer Component above the Change Mode Link



5. 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 **Resource Palette** from the View menu.

6. Under My Catalogs, open the **WebCenter Services Catalog**, then expand **Task Flows**.
7. Locate the **Search Toolbar** task flow (Figure 5–6).

Figure 5–6 Search Toolbar Task Flow in the WebCenter Services Catalog

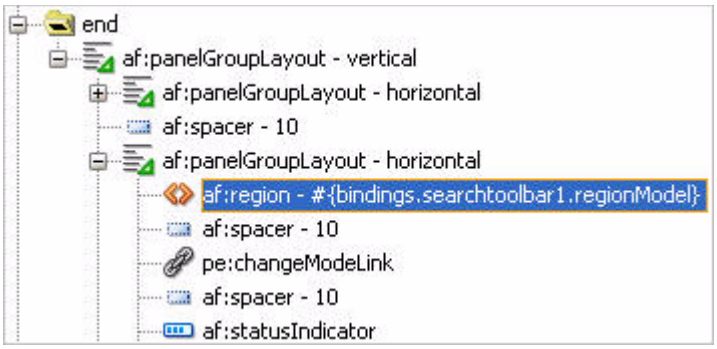


Drag and drop this task flow onto the Panel Group Layout then choose **Region** from the context menu. If you are prompted to add the appropriate libraries, choose **Add Libraries**.

8. In the Structure window, move the **Search Toolbar** task flow (`af:region - #{bindings.sarchtoolbar1.regionModel}`) above the Spacer component you just added (Figure 5–7).

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 5–7 Search Toolbar in the Structure Window



9. Run the page to your browser and log in as user `Lisa/welcome1` to see how it looks at runtime. Figure 5–8 shows the Search toolbar next to the Edit (Change Mode) link. You can test the Search toolbar later in the chapter, after you add content on which you can perform a search.

Figure 5–8 Search Toolbar on MyPage at Runtime

10. Return to JDeveloper.

For more information about the Search service, including other Search task flows you can use, see Chapter 24, "Integrating the Search Service" in *Oracle Fusion Middleware Developer's Guide for Oracle WebCenter*.

Now that you have added a Search toolbar to our application, continue to the next step to add the Documents service.

Step 2: Create a Connection for the Documents Service

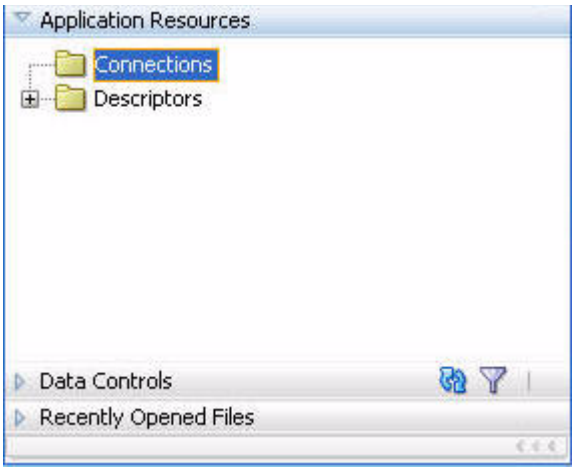
With most services in the WebCenter Framework, you perform a few basic steps: configure the service with our application, add the service task flow to our page, then run the page and customize or use the service at runtime.

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."

To create a connection to our content repository:

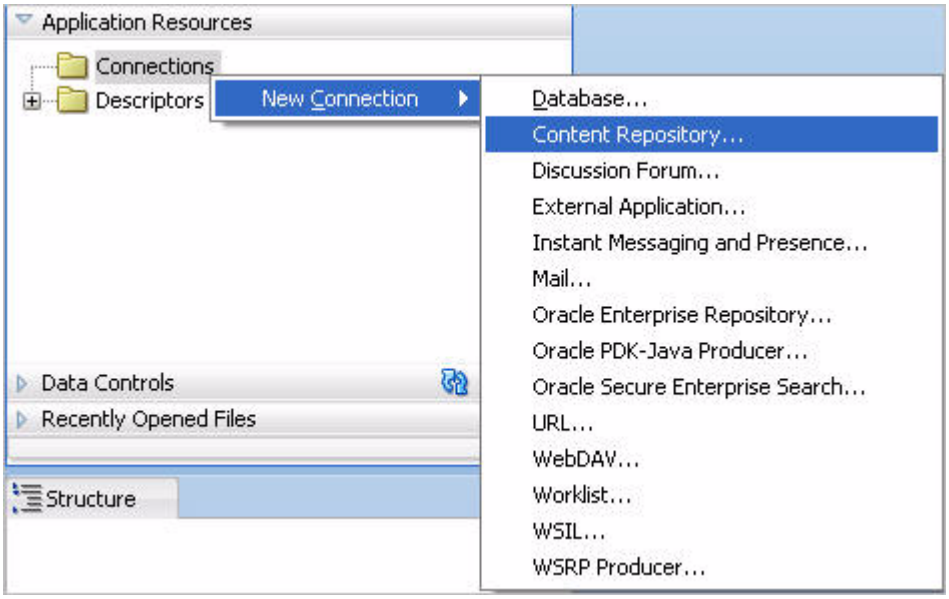
1. In Oracle JDeveloper, in the Application Navigator, locate the Application Resources node for MyTutorialApplication and expand it. Notice the **Connections** folder here (Figure 5–9).

Figure 5–9 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**, then choose **Content Repository...** (Figure 5–10).

Figure 5–10 Choosing the New Content Repository Connection Menu Option



4. In the Create Content Repository Connection dialog 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.

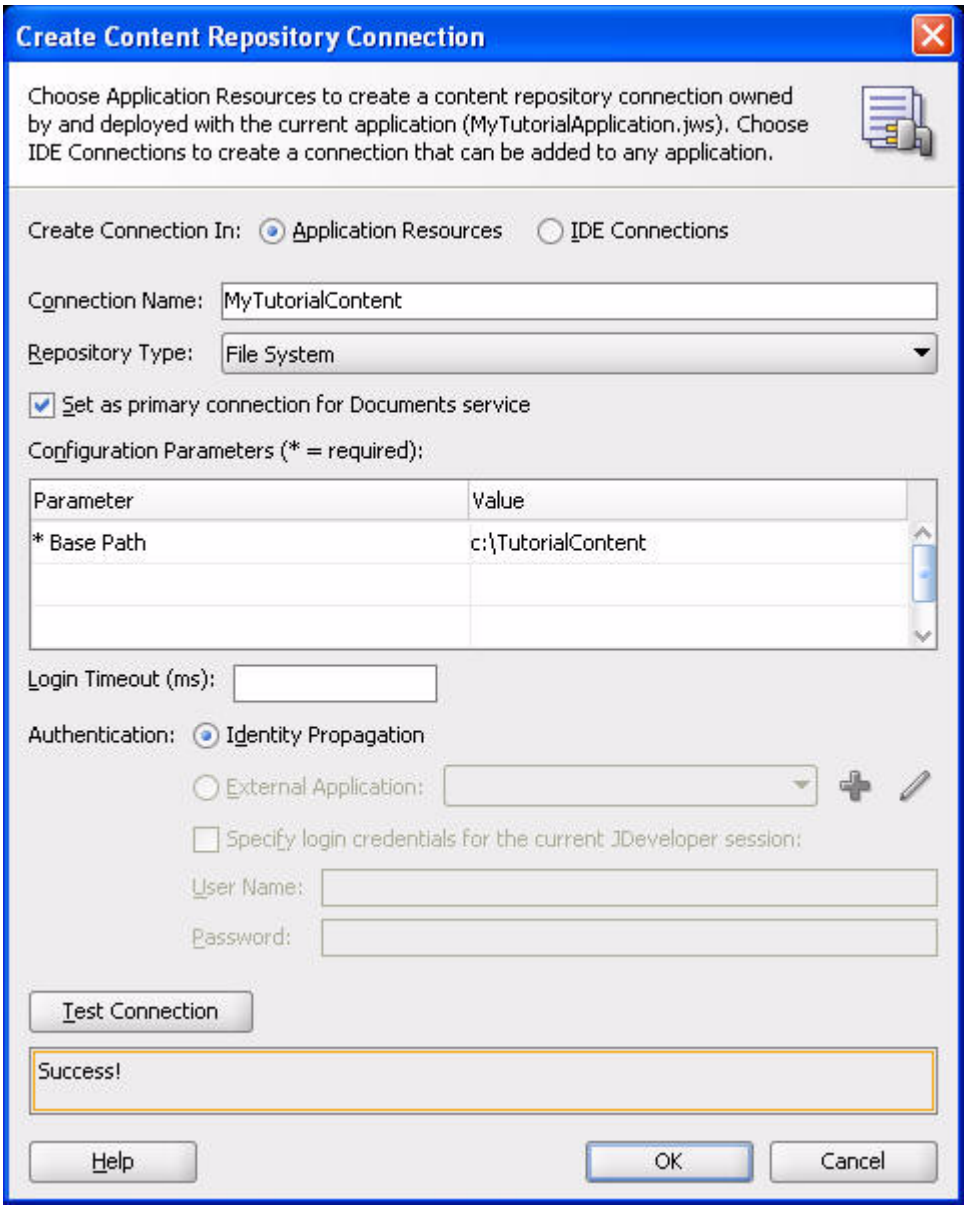
   Select **Application Resources**.

> **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.

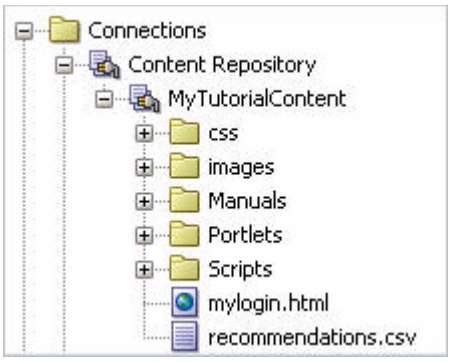
   If you did not choose this option and created a connection to another content repository, Oracle WebCenter Framework would automatically set the first connection you created as the active connection.
8. Under Configuration Parameters, let's set the Base Path to the location where you downloaded the Tutorial 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 5–11.

Figure 5–11 Create Content Repository Connection



10. Click **OK**.
11. In the Application Navigator, in the Application Resources panel, you will see your new connection, as shown in Figure 5–12.

Figure 5–12 New Connection in the Application Resources Panel



For more information about creating a connection for the Documents service, see Chapter 14, "Integrating the Documents Service" in *Oracle Fusion Middleware Developer's Guide for Oracle WebCenter*.

Step 3: Add the Documents - Document Manager Task Flow to Your Application

Oracle WebCenter Framework enables content integration by:

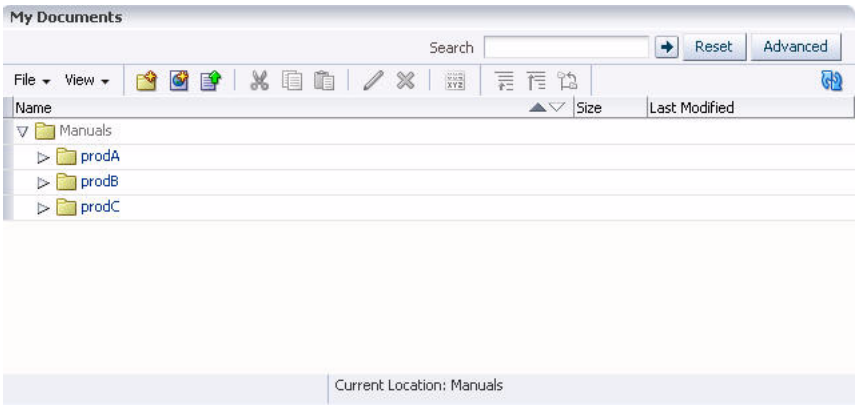
- Content Repository data controls: enable read-only access to a content repository, and maintain tight control over the way the content displays in the application.
- Documents service: enable users to view and manage documents in your organization's content repositories.

Both methods of integration use content repository connections, which you created in the previous step. In this Tutorial, we will 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 14, "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 Documents - Document Manager task flow. This 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 5–13).

Figure 5–13 Document Manager View of Sample Folders at Runtime

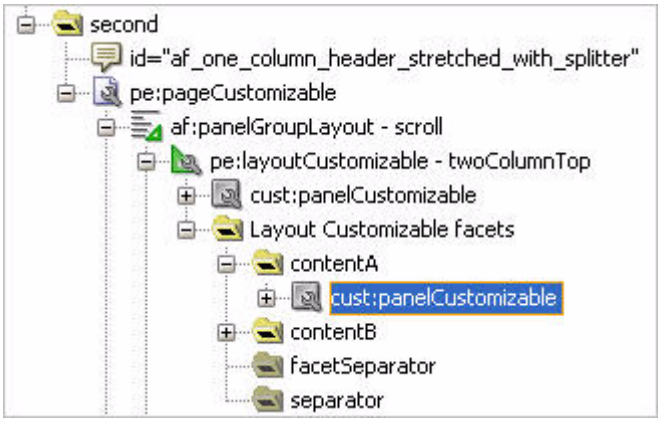


To add the Documents - Document Manager task flow to your application at design time in Oracle JDeveloper:

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 Documents - Document Manager 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 5–14). Under the **second** facet navigate to **pe:pageCustomizable** then **af:panelGroupLayout - scroll**, **Layout Customizable facets**, then **contentA**. 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 5–14 Location Where We Will Add the Documents - Document Manager Task Flow



4. While the Panel Customizable is selected, set the **Layout** property to **stretch**.

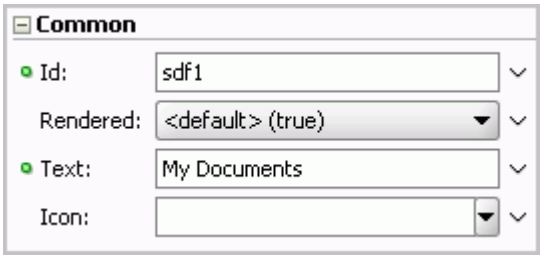
5. Add a chrome, so that at runtime, the layout shows a frame around the list of documents.

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

6. Under Layout, drag and drop **Show Detail Frame** onto the **cust:panelCustomizable** that you selected (as shown in Figure 5–14).

7. While the Show Detail Frame is selected, in the Property Inspector, under Common, change the **Text** property to `My Documents`, as shown in Figure 5–15. Notice also that the frame is also selected in the Design view of your page.

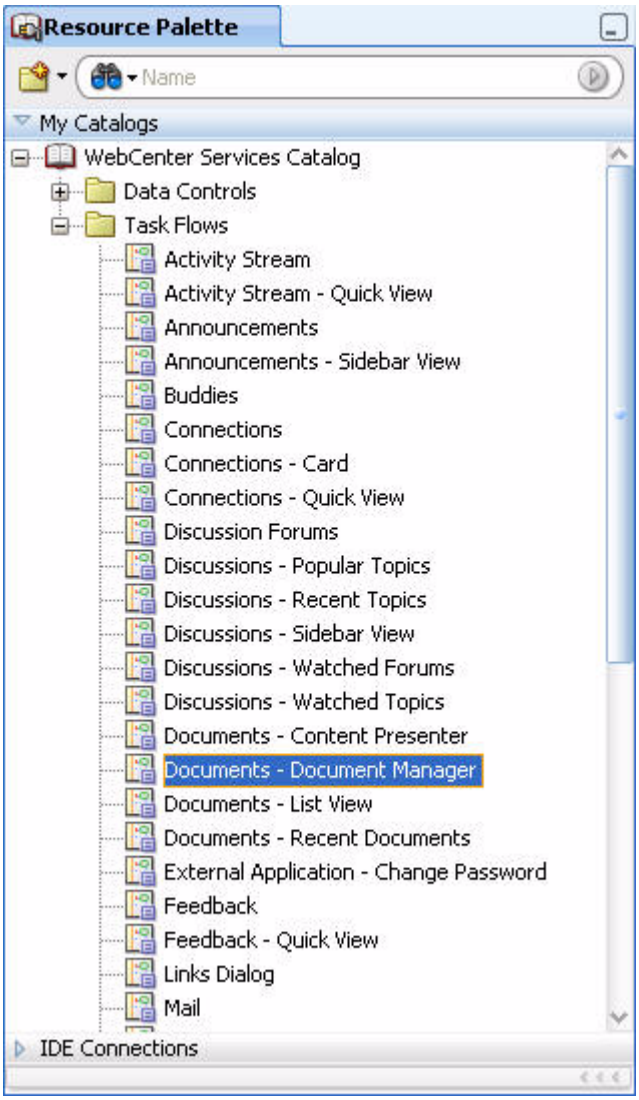
Figure 5–15 Changing the Title of the Show Detail Frame



8. You can add the Documents - Document Manager task flow to your application in one of two ways: either by dragging and dropping the connection you created in "Step 2: Create a Connection for the Documents Service", called `MyTutorialContent`, onto the Show Detail Frame, then choosing **Document - Document Manager** from the context menu, or by using the Resource Palette. For the purposes of this Tutorial, you can use the Resource Palette.

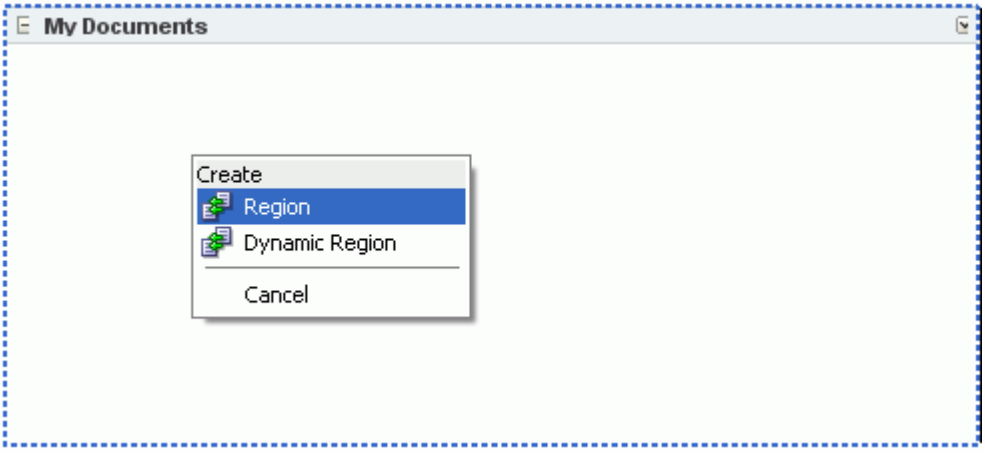
   In the Resource Palette, under WebCenter Services Catalog, open the **Task Flows** folder and locate **Documents - Document Manager**. Figure 5–16 shows the task flow in the WebCenter Services Catalog.

Figure 5–16 Documents - Document Manager Task Flow in the Resource Palette



9. Drag and drop the **Documents - Document Manager** 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.
10. In the context menu that displays, click **Create Region**, as shown in Figure 5–17.

Figure 5–17 Dropping the Documents - Document Manager Task Flow onto the Page in the Design View



11. If the Confirm Add ADF Library dialog displays, click **Add Library**.
12. The Edit Task Flow Binding dialog displays. In this dialog, 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.

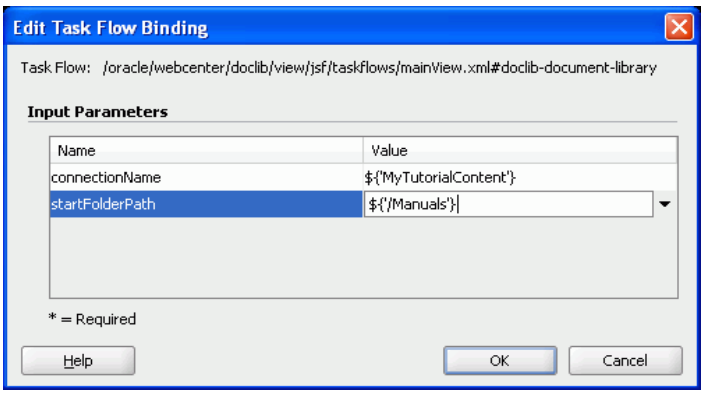
    Note: If you do not enter a connection name here, the content repository connection where you selected **Set as primary connection for Documents service** is the active connection.

    ---

13. 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.

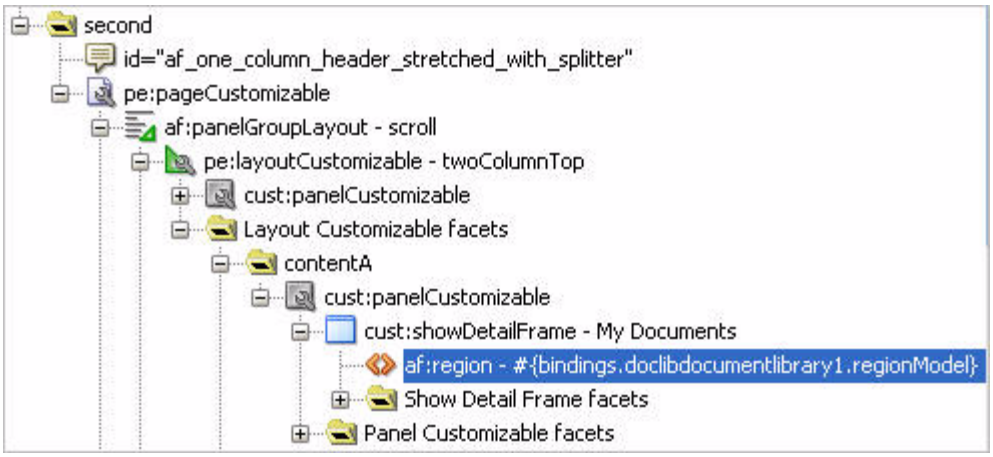
    Click the **startFolderPath** parameter and enter `${'/Manuals'}` next to it, then click **OK**. Figure 5–18 displays the Edit Task Flow Binding dialog with the appropriate values.

Figure 5–18 Edit Task Flow Binding Dialog



The Structure window shows the new task flow with the `af:region` tag, as shown in Figure 5–19.

Figure 5–19 Documents - Document Manager Task Flow in the Structure Window



14. In the Application Navigator, right-click **MyPage.jspx** and choose **Run**.
15. Log in as user `Lisa` with the password `welcome1`. Because the content repository used with the Documents - Document Manager task flow is local, this user has access to the content. However, if you used a secure content repository, such as Oracle Content Database or Oracle Content Server, you would need to ensure that the user credentials for your WebCenter application match those of a user on the secure content repository.

For more information about the Documents - Document Manager task flow, as well as other Documents service task flows, see Chapter 14, "Integrating the Documents Service" in the *Oracle Fusion Middleware Developer's Guide for Oracle WebCenter*.

In the next step, we will examine the Documents - Document Manager task flow at runtime.

Step 4: Browse Documents at Runtime

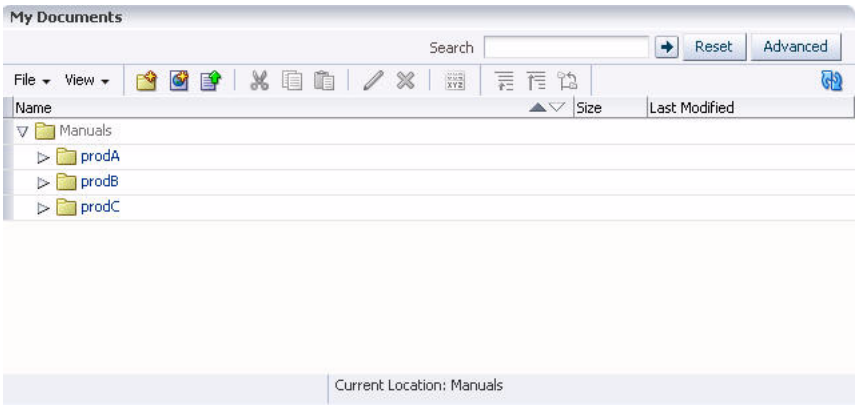
Now that you've added the task flow to your application, you can see the documents in your content repository, or "document library."

This step covers just a few of the basic tasks you can perform with the Documents service at runtime. For more information on using the Documents service, see Chapter 15, "Working with the Documents Service" in *Oracle Fusion Middleware User's Guide for Oracle WebCenter*.

To browse your documents at runtime:

1. After you run your application to your browser and log into the application, a list of your files display (Figure 5–20). The Documents service's Document Manager view shows a document library based on the file system content repository you created on your local drive. The difference between this view and other views (such as the Content Presenter or List views) provided by the Documents service is that you can delete, modify, and edit your files (that is, *manage* files) in the document library as you would on your file system.

Figure 5–20 Documents - Document Manager View at Runtime

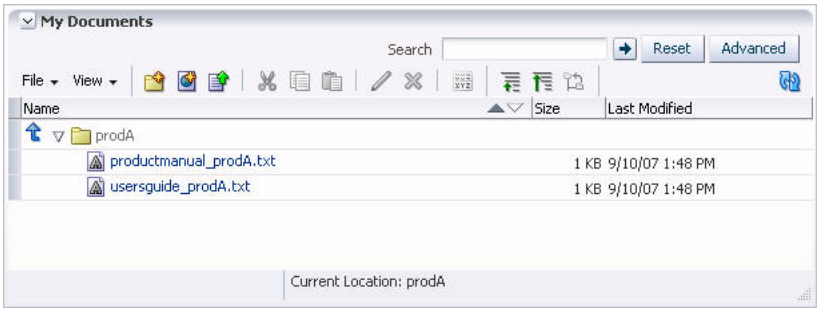


Let's examine this view. As you can see, the service shows by default the names of the folders: `prodA`, `prodB`, `prodC`. This information all comes from your file system and let's you quickly show your content without much coding.

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 5–21 Documents - Document Manager Task Flow Showing the Contents of a Folder

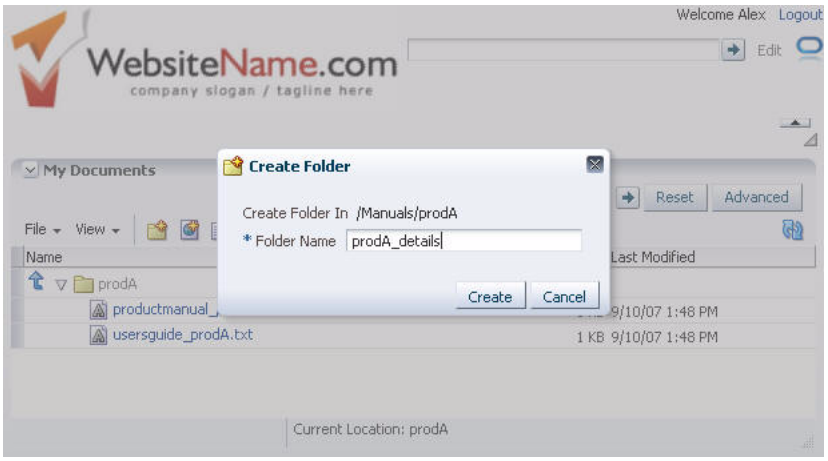


3. Click the **Create New Folder** icon in the toolbar (Figure 5–22).

Figure 5–22 Create New Folder Icon in the Toolbar

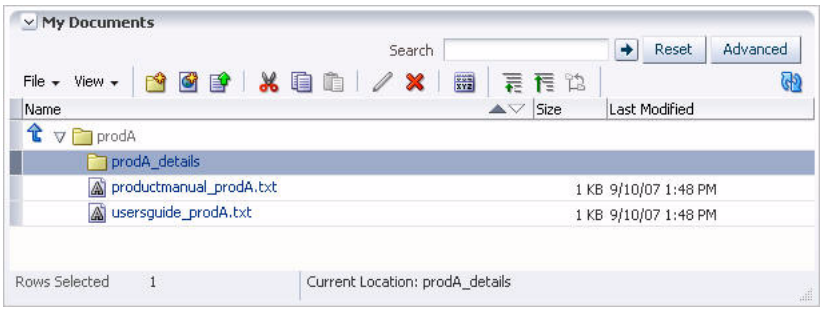
4. In the Create Folder dialog, enter a folder name, such as `prodA_details` (Figure 5–23).

Figure 5–23 Documents - Document Manager Task Flow with the Create Folder Dialog



5. Click **Create**. Figure 5–24 shows the new folder in your browser.

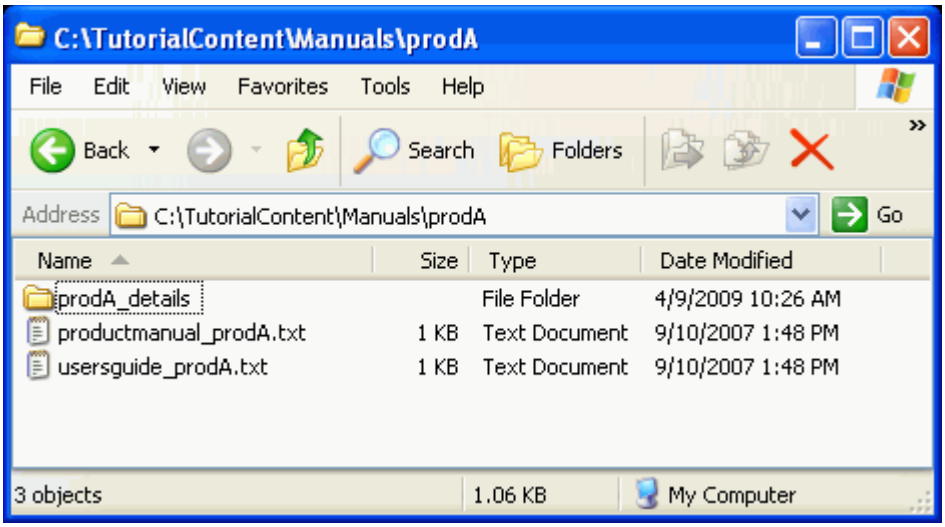
Figure 5–24 prodA_details Folder in the Documents - Document Manager View at Runtime



6. Since you are using your own file system as your content repository, any actions you perform at runtime also affect the content repository itself. Take a look at the content repository, in this case on your file system.

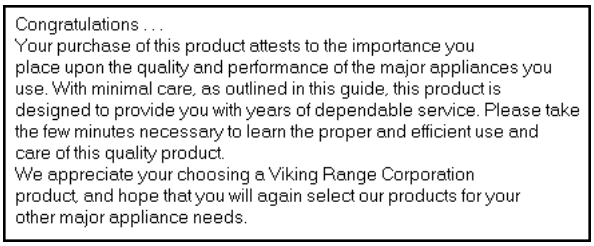
   Notice the `prodA_details` folder also exists in your `C:\TutorialContent\Manuals\prodA` directory (Figure 5–25).

Figure 5–25 New Folder in the File System



7. Return to your browser to view the Document Library at runtime.
8. Click the file **productmanual_prodA.txt**. Notice that a new browser tab or window displays with the contents of the text file (Figure 5–26).

Figure 5–26 Text File in a Browser Window



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.

9. Return to Oracle JDeveloper.

As you can see, the Documents service provides a quick and easy way for you to display content in a meaningful way and with functionality. You can learn more about the Documents service task flows in Chapter 14, "Integrating the Documents Service" in *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 13 "Integrating Content" in *Oracle Fusion Middleware Developer's Guide for Oracle WebCenter*.

Step 5: Create a Database Connection to the WebCenter Schema

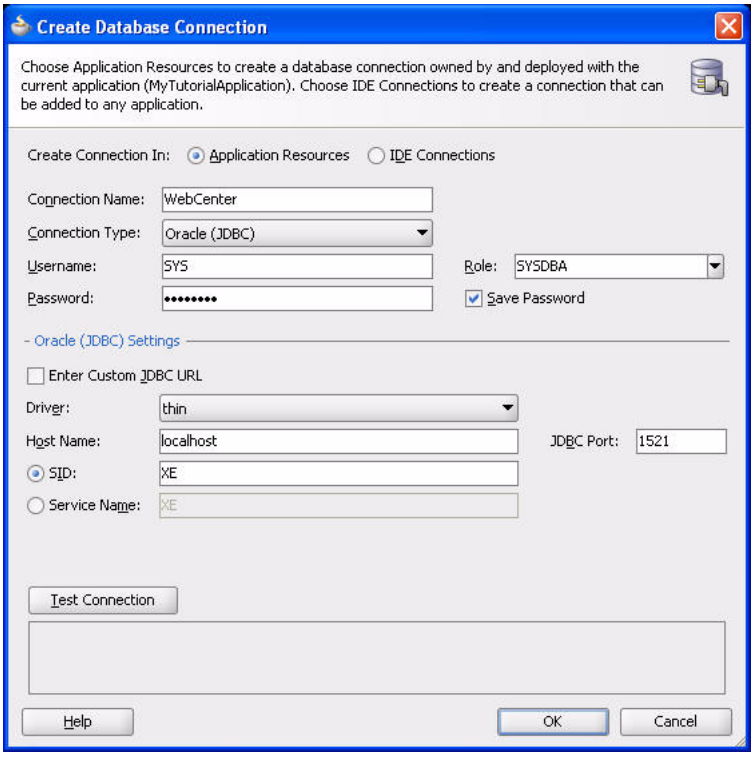
Next, we will add the Tags, People Connections, and Links services. To use these three services, you must have access to the WebCenter schema. You installed this schema in Chapter 2, "Preparing for the Tutorial." In this section, you will create a connection to the database containing this schema, so that these three services can use this schema within the application. The schema will be used to store information about the services.

To learn more about setting up your application for consuming services, see Chapter 4, "Preparing Your Application for Oracle WebCenter Services" in the *Oracle Fusion Middleware Developer's Guide for Oracle WebCenter*.

To create the database connection to the WebCenter schema:

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 5–27). 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 5–27 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 are not sure what you entered when you installed it).
5. Click **OK**.

Now that you have created the database connection, you can add the Tags, People Connections, and Links services 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.

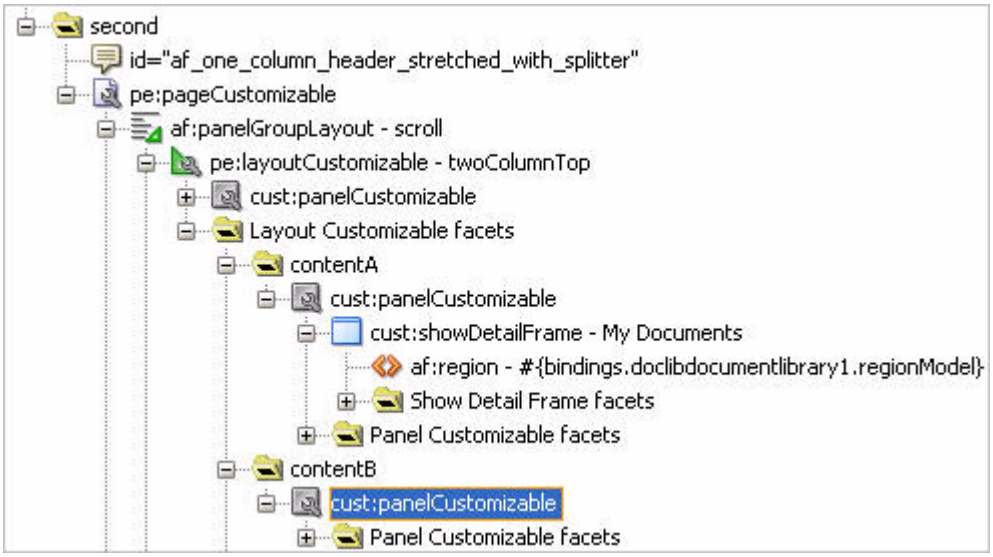
A "tag cloud" is a visual illustration of all the tags for the application, making it easy for you and your users to identify the tags used in the application. You can then search for a tag in your application to locate any item that has been associated with that tag.

In this section, you will add the Tags service to our application and display it on your page. You will add a Tag Cloud so that you can visualize the tags and enable you and your users to add new tags to the application.

To add the Tags service:

1. In the Structure window for MyPage, navigate to the **Panel Customizable** within the facet `contentB` of the Layout Customizable (Figure 5–28).

Figure 5–28 Panel Customizable in contentB

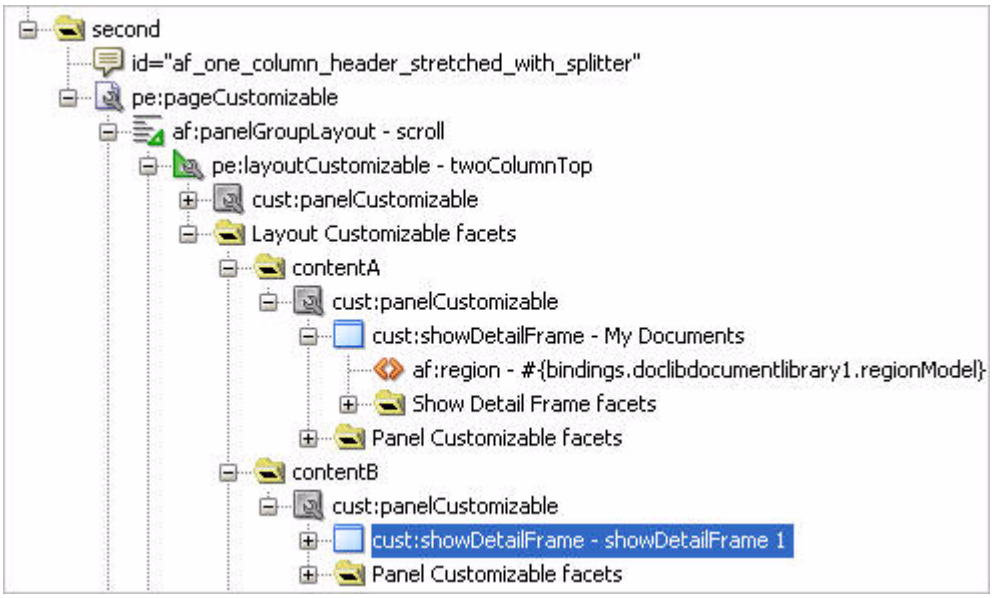


Note: As previously mentioned, you can use the pushpin in the Structure window to freeze the current view. For this step, you should click **MyPage** in the Design view, then ensure the pushpin is in the "freeze" position (pressed).

2. Set the **Layout** property for the Panel Customizable to **stretch**.
3. In the Component Palette, choose **Oracle Composer** from the list.

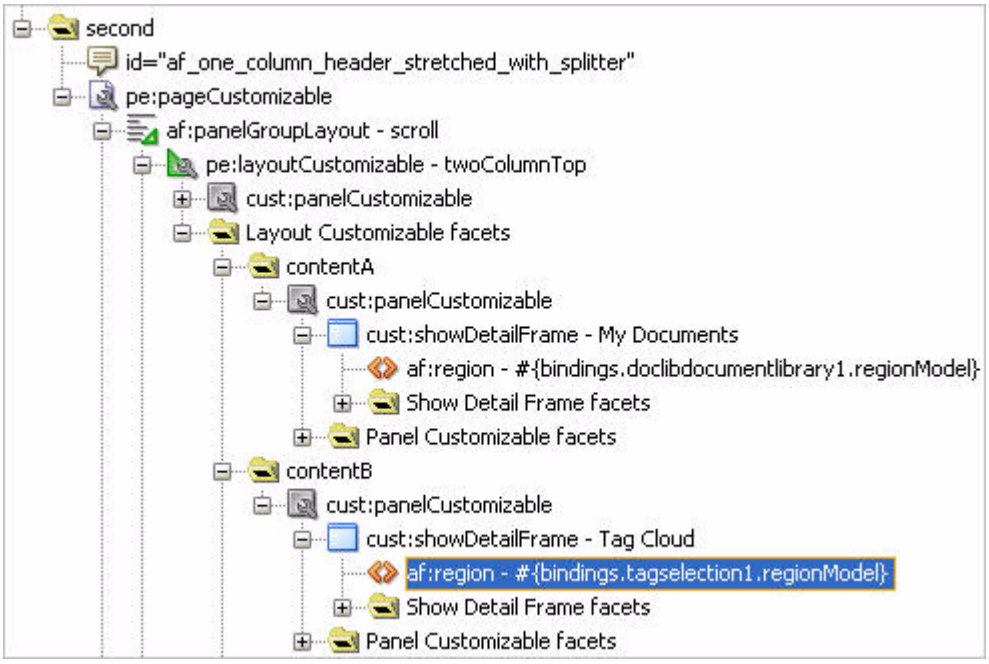
4. Under Layout, drag and drop a **Show Detail Frame** onto the Panel Customizable, as shown in Figure 5–29.

Figure 5–29 Second Show Detail Frame



5. While the Show Detail Frame is selected, in the Property Inspector, under Common, change the **Text** property to `Tag Cloud`.
6. 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 Tag Cloud Show Detail Frame.
7. In the Create Region context menu, choose **Create Region**. The Tagging - Tag Cloud task flow displays in the Structure window, as shown in Figure 5–30.

Figure 5–30 Tagging - Tag Cloud Task Flow in the Tag Cloud Show Detail Frame



8. Right click **MyPage** in the Design view and choose **Run**, then log into the application with the username `Lisa` and password `welcome1`.

For more information about the Tag Cloud and the Tags Service, refer to Chapter 23, "Integrating the Tags Service" in the *Oracle Fusion Middleware Developer's Guide for Oracle WebCenter*.

We will examine the Tags service at runtime in the next step.

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

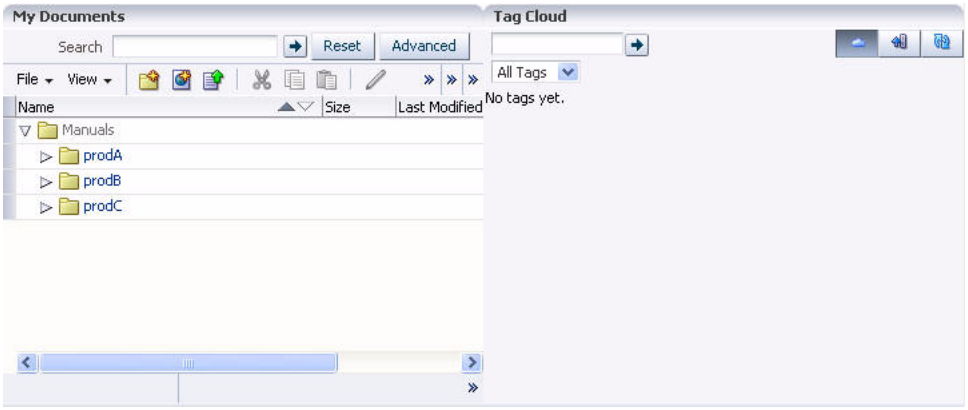
After you add the Tagging - Tag Cloud task flow to your page, the Tag Cloud automatically populates with the tags in the WebCenter schema. If you installed the WebCenter schema for the purposes of this Tutorial, you will not see any tags in the Tag Cloud. This step shows you the Tag Cloud at runtime, as well as a few basic steps you can perform with the Tags Service. You can learn more about using the Tags Service at runtime in Chapter 26, "Working with the Tags Service" in *Oracle Fusion Middleware User's Guide for Oracle WebCenter*.

To use the Tags service at runtime:

1. In your browser, you should now see the Tag Cloud displaying below the Connections view. 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 5–31 shows how MyPage looks at runtime.

Figure 5–31 Tag Cloud at Runtime

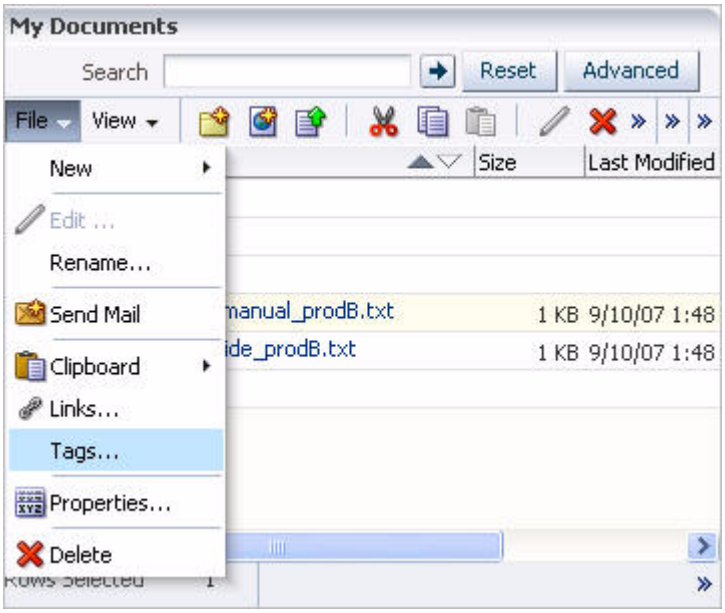


2. Once you add the Tags service to your application, you can use it with the Document Library service. Let's see how we can add tags to a document in our document library.

   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 will just display the text file in a new browser window as you did in "Step 4: Browse Documents at Runtime".

3. In the Documents menu, choose **Tags...** from the File menu. (Figure 5–32).

Figure 5–32 Choosing the Tags Menu Option in the Documents - Document Manager View



4. The Tag this Document dialog displays. Add a few tags, such as (Figure 5–33):

   ```
   productb guide prodb
   ```

Figure 5–33 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_prodA.txt**.
6. In the Tag this Document dialog, enter a few tags, then save your changes:

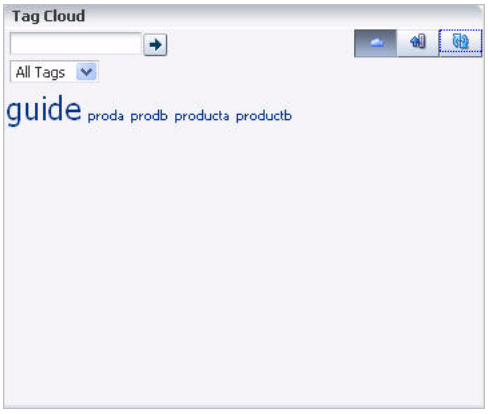
   ```
   producta guide proda
   ```

7. Click the **Refresh** icon to the right of the Tag Cloud title bar (Figure 5–34).

Figure 5–34 Refresh Tag Cloud Icon

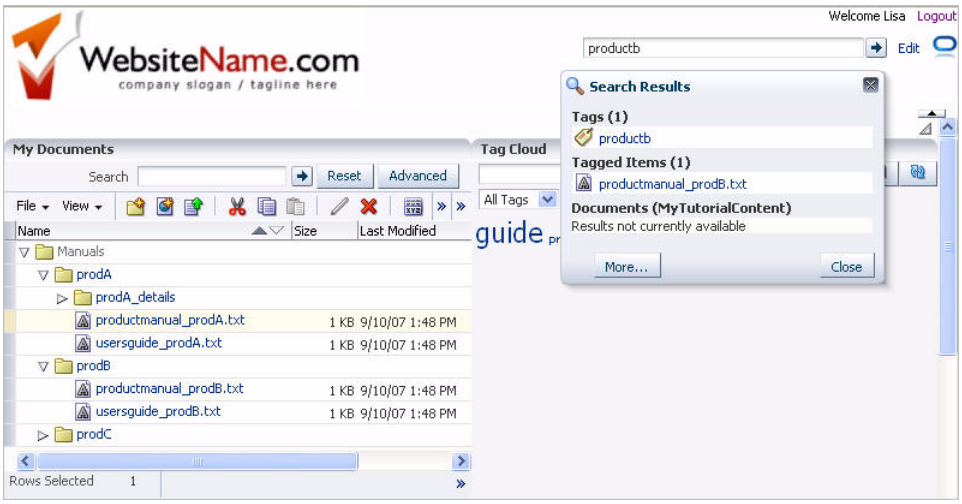
Notice that the Tag Cloud now displays your new tags, for example **producta**, **productb**, **guide**, and **prodb** as shown in Figure 5–35.

Figure 5–35 New Tags in the Tag Cloud



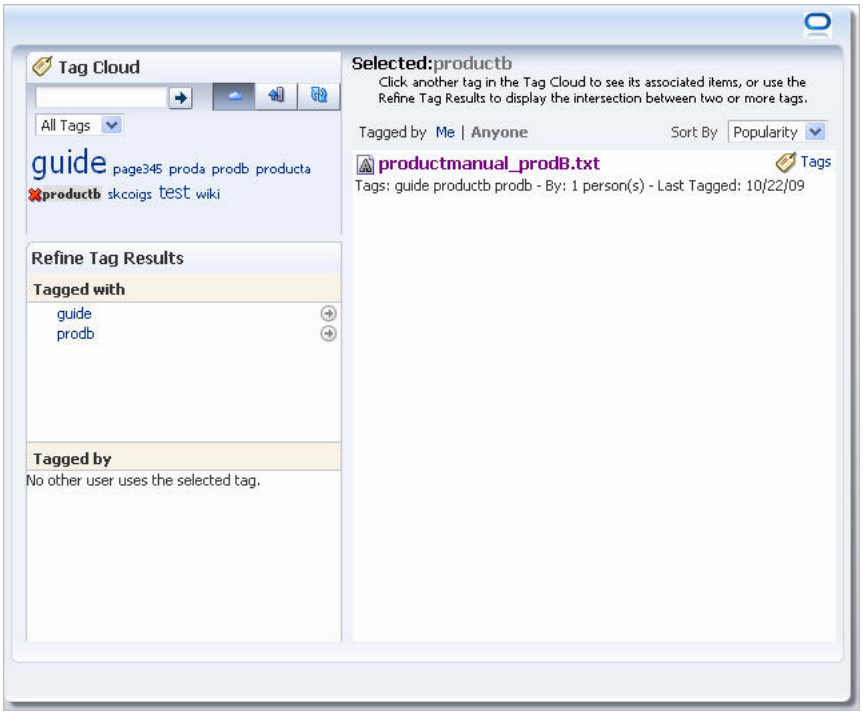
8. Check out how tags help you locate items in your application. In the Search toolbar at the top of the page, enter `productb`, then click the arrow icon. You will see the Search Results return the tag for `productb` (Figure 5–36).

Figure 5–36 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 5–37).

Figure 5–37 Tag Center



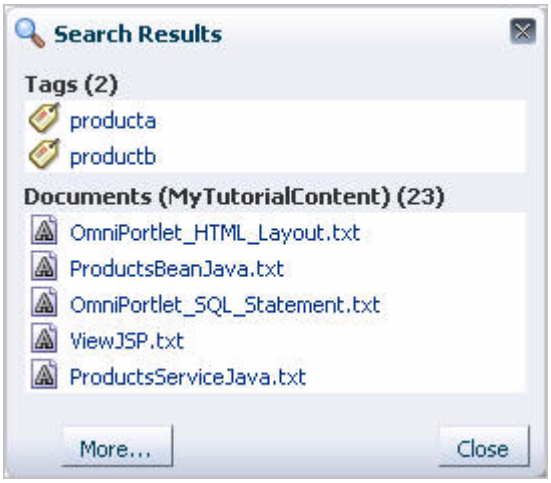
The Tag Center displays the Tag Cloud, all the results from your search that are associated with the tag `productb`, including other advanced search options. You can close this window after you are done viewing the different options and return to your application.

10. If you are interested, add a few more tags to see how the Tag Cloud changes. Otherwise, you can move on to the next step, "Step 8: Add the People Connections Service to Your Application".

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 5–38.

Figure 5–38 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.

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

    Figure 5–39 shows the search results. Notice that the *tagged item*, `productmanual_prodA.txt` displays. Because the item was tagged with "producta," our search returned the specific item.

Figure 5–39 Search Results for producta

Now that you have added the Search toolbar, Documents - Document Manager view, and the Tag Cloud to your application, you can see how tagging and search work together to enable you to easily locate files in a content repository or document library.

Next, let's see how we can add a social networking feature to the application.

Step 8: Add the People Connections Service to Your Application

The People Connections service provides social networking tools for creating, interacting with, and tracking the activities of one's enterprise connections. Its features enable users to manage their personal profiles, access the profiles of other users, provide ad hoc feedback, post messages, track activities, and connect with others. This step introduces you to using the task flows associated with the People Connections service.

In this Tutorial, you will take advantage of four People Connections task flows:

- the *Connections* task flow, which enables you to invite and accept invitations from other application users to become part of your social network
- the *Activity Stream* task flow, which displays recent activities within services across your application -- in this case, Documents and People Connections
- the *Message Board* task flow, which displays messages from the integrated message board
- the *Profile* task flow, which provides users with a variety of views into their own and other users' personal profile information, such as their email address, phone number, and so on.

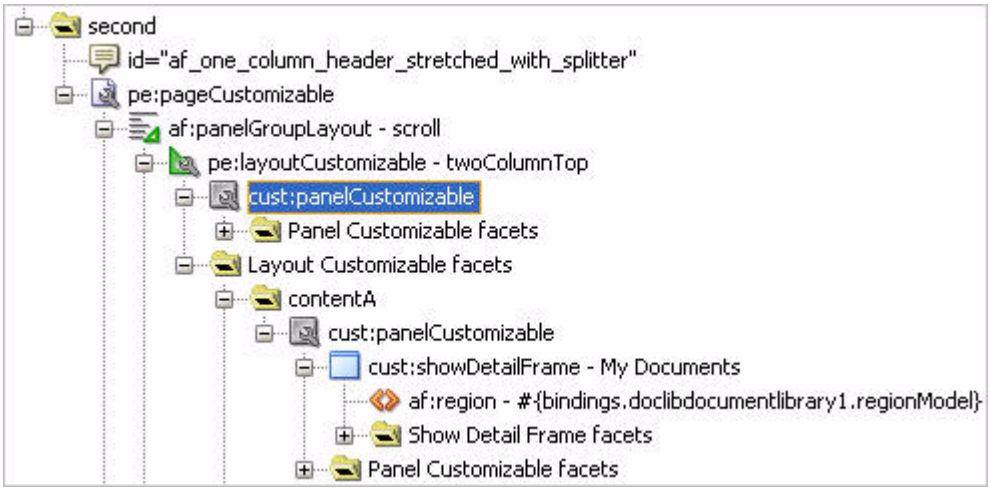
After you add these task flows, you can see how they work at runtime in the next step.

To add the People Connections service:

1. Ensure that you have created the database connection to the WebCenter schema per "Step 5: Create a Database Connection to the WebCenter Schema".
2. In the Structure window for MyPage, navigate to the **second** facet, where you added the Documents and Tags task flows, then find the Layout Customizable, and the first Panel Customizable, as shown in Figure 5–40.

> **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).

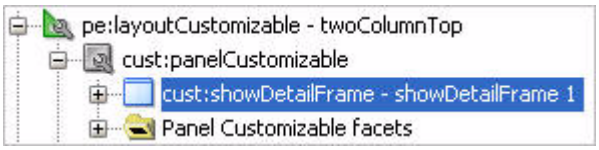
Figure 5–40 First Panel Customizable in the Layout Customizable



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

4. Under Layout, drag and drop a **Show Detail Frame** onto the Panel Customizable, as shown in Figure 5–41.

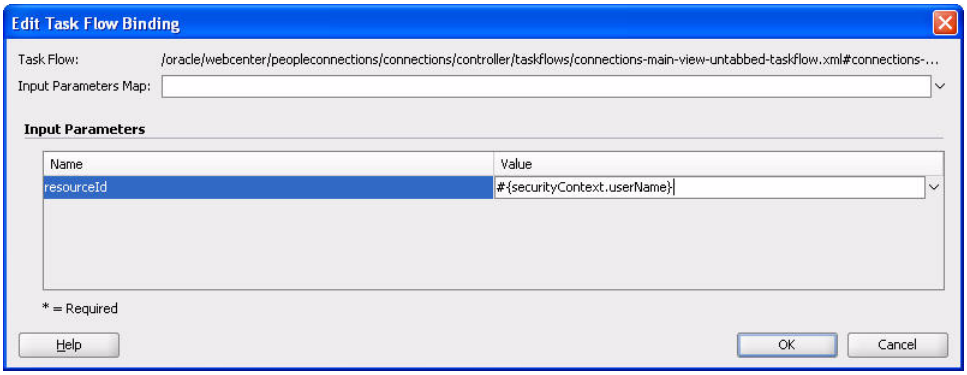
Figure 5–41 Panel Customizable Containing the Show Detail Frame



5. While the Show Detail Frame is selected, in the Property Inspector, under Common, change the **Text** property to `People Connections`.
6. In the Resource Palette, under WebCenter Services Catalog, open the **Task Flows** folder and locate the **Connections** task flow, then drag and drop it onto the Show Detail Frame you just added.
7. In the Create Region context menu, choose **Create Region**.
8. In the Edit Task Flow Binding dialog, set the **resourceId** parameter to the following value (Figure 5–42):

```
#{securityContext.userName}
```

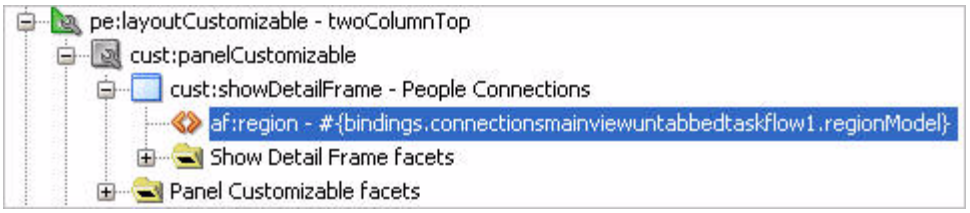
Figure 5–42 Edit Task Flow Binding Dialog - Connections



This expression tells the Connections task flow to display all the connections of the currently logged in user. You can alternatively use the Expression Builder to set this value. To do so, click the down arrow next to the field, then choose **Expression Builder**. In the Expression Builder dialog, expand **ADF Bindings** and **securityContext**. select **userName**, then click **OK**.

9. Click **OK**. The Connections task flow displays in the Structure window, as shown in Figure 5–43.

Figure 5–43 Connections Task Flow in the Structure Window

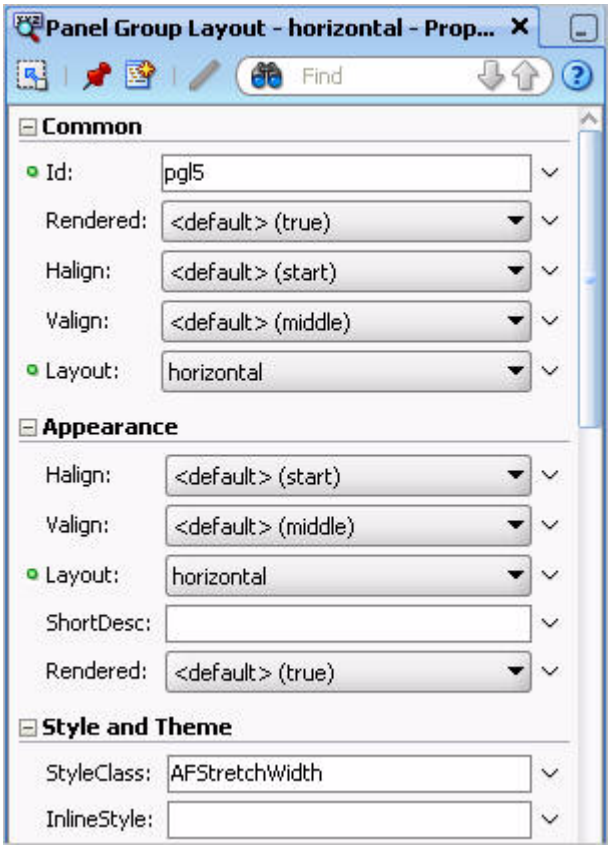


10. From the Component Palette, drag and drop a **Panel Group Layout** ADF Faces component onto the Panel Customizable containing the People Connections Show Detail Frame, as shown in Figure 5–44.

Figure 5–44 New Panel Group Layout

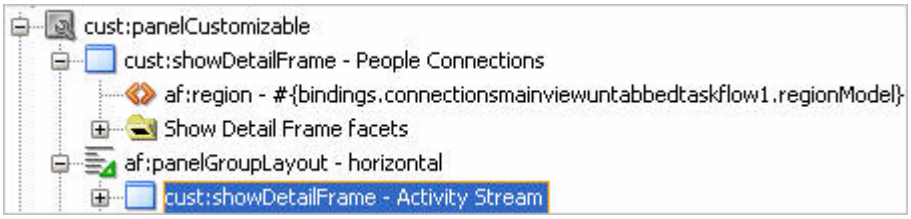
11. In the Property Inspector for the Panel Group Layout, under Common, set the **Layout** property to **horizontal**.
12. Under Style and Theme, set the **Style Class** property to `AFStretchWidth`. Figure 5–45 shows the Property Inspector with the two properties updated.

Figure 5–45 Properties for the Panel Group Layout



13. Drag and drop a **Show Detail Frame** (Oracle Composer component) onto the Panel Group Layout.
14. While the Show Detail Frame is selected, in the Property Inspector, under Common, change the **Text** property to `Activity Stream`. Figure 5–46 shows the new Show Detail Frame in the Structure window.

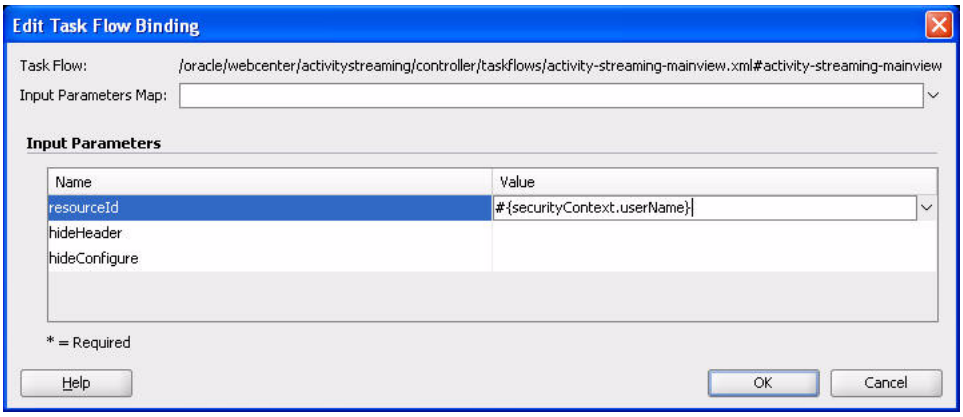
Figure 5–46 Activity Stream Show Detail Frame



15. In the Resource Palette, under WebCenter Services Catalog, open the **Task Flows** folder and locate the **Activity Stream** task flow, then drag and drop it onto the Show Detail Frame you just added.
16. In the Edit Task Flow Binding dialog, set the **resourceId** parameter to the following value (Figure 5–42):

```
#{securityContext.userName}
```

Figure 5–47 Edit Task Flow Binding Dialog - Activity Stream

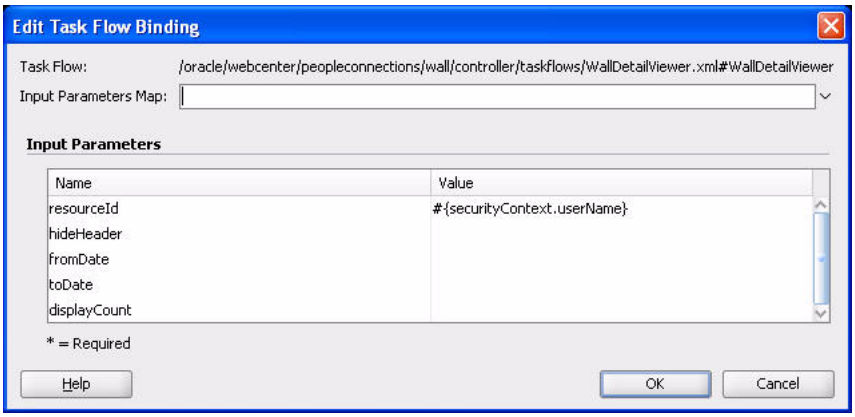


This expression tells the Activity Stream task flow to display all the recent activities of the currently logged in user. You can alternatively use the Expression Builder to set this value. To do so, click the down arrow next to the field, then choose **Expression Builder**. In the Expression Builder dialog, expand **ADF Binding**s and **securityContext**. select **userName**, then click **OK**.

17. Add another **Show Detail Frame** directly below the one containing the Activity Stream task flow. You can do this by dropping the Show Detail Frame onto the `af:panelGroupLayout - horizontal` containing the Activity Stream Show Detail Frame.
18. While the Show Detail Frame is selected, in the Property Inspector, under Common, change the **Text** property to `Messages`.
19. From the Resource Palette, drag and drop the **Message Board** task flow onto the page. This task flow provides users with the main view of Message Board messages and a means of adding, viewing, and managing Message Board messages.
20. In the Create Region context menu, choose **Create Region**.
21. In the Edit Task Flow Binding dialog, set the **resourceId** parameter to the following (Figure 5–48):

```
#{securityContext.userName}
```

Figure 5–48 Edit Task Flow Binding Dialog - Message Board



22. Click **OK**.

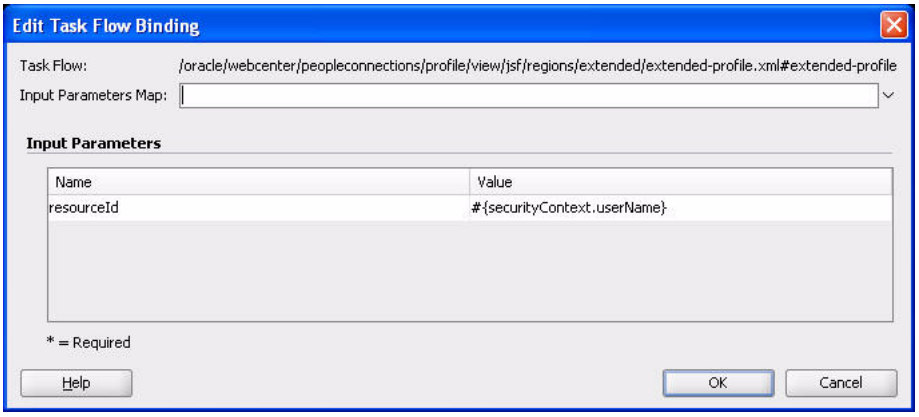
23. Drop a Show Detail Frame onto the Panel Customizable containing the People Connections Show Detail Frame and set the **Text** property to `My Profile`. This Show Detail Frame should be at the same level in the structure as the horizontal Panel Group Layout containing the Activity Stream and Message Board Show Detail Frame components.

24. From the Resource Palette, drag and drop the **Profile** task flow onto the My Profile Show Detail Frame.

25. In the Edit Task Flow Binding dialog, set the **resourceId** parameter to the following (Figure 5–49):

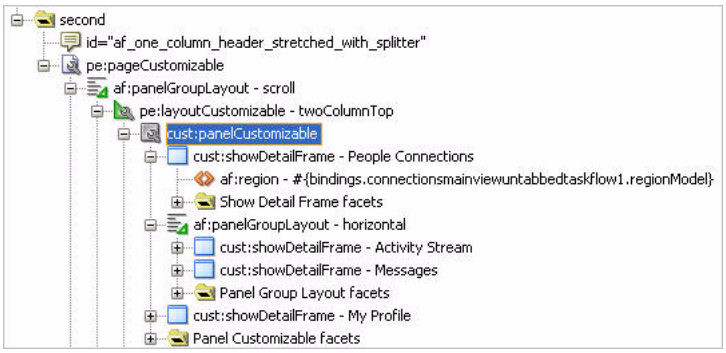
    ```
    #{securityContext.userName}
    ```

Figure 5–49 Edit Task Flow Binding Dialog - Profile



26. Click **OK**. The structure should now look like Figure 5–50.

Figure 5–50 People Connections, Activity Stream, Message Board, and Profile Task Flows in the Structure Window



27. Since you already have security configured, the task flows you just added by default have security enabled on them for the `authenticated-role`.

    Right click **MyPage** in the Design view and choose **Run**, then log in as user `Lisa` with the password `welcome1`.

For more information about the People Connections service, see Chapter 20, "Integrating the People Connections Service" in the *Oracle Fusion Middleware Developer's Guide for Oracle WebCenter*.

We will examine the People Connections service at runtime in the next step.

Step 9: Use the People Connections Service in Your Application at Runtime

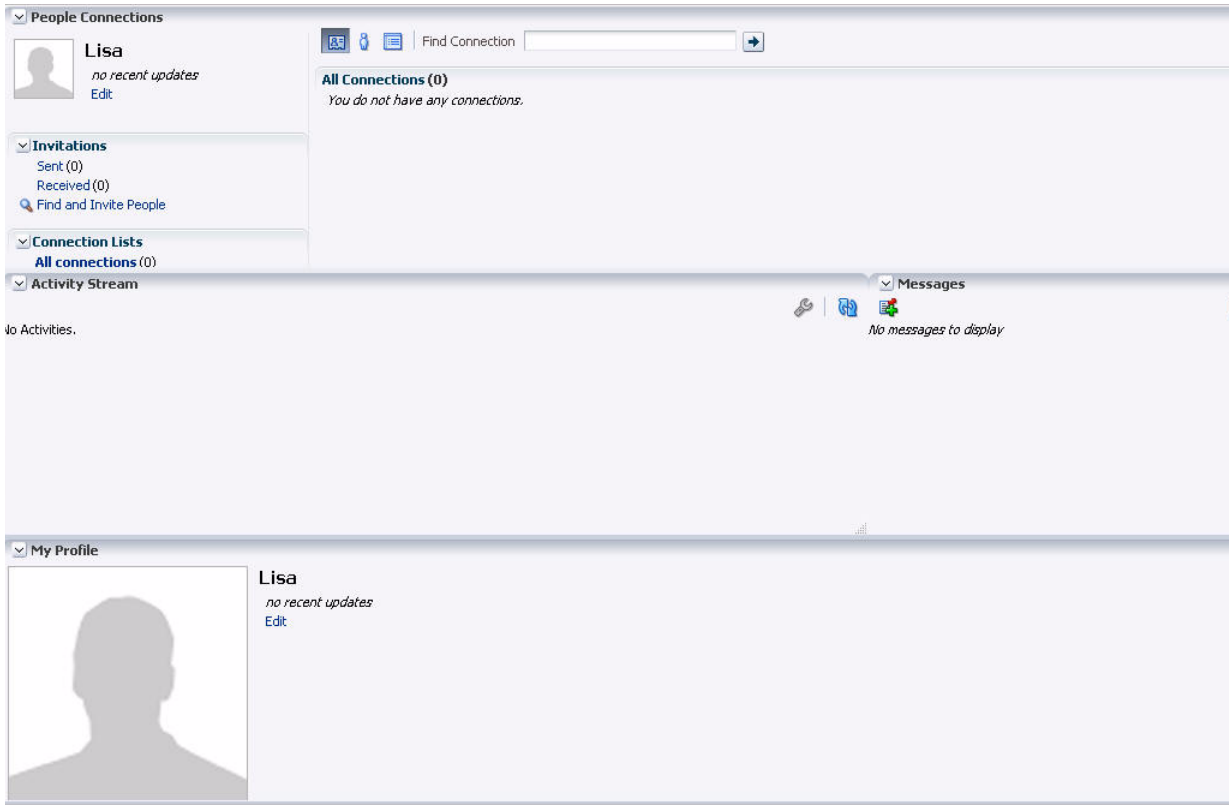
In this step, you will use the three users you created in Chapter 4, "Adding Security to Your Application" to test how the service would work in a production environment. Instead of three different people adding each other to one another's networks, you will log in as the three different test users (Alex, Dan, and Lisa) to see how the People Connections service works at runtime.

While this step shows you a few of the basic tasks you can perform using these task flows, you can learn more about using the service at runtime in Chapter 22, "Working with the People Connections Service" in the *Oracle Fusion Middleware User's Guide for Oracle WebCenter*.

To use the People Connections service in your application:

1. After you run the page and log into the application as user `Lisa`, you should now see the Activity Stream view, the Connections view and the Message Board view in your application.

Figure 5–51 People Connections Service on MyPage



2. Notice that the Activity Stream and Message Board task flows currently display no updates. As you update the application either by adding a connection or updating

a document in the Documents - Document Manager view, notice how the information in these two views update.

3. Let's update Lisa's profile by uploading an image. In the My Profile Show Detail Frame, scroll down below the blank profile image to find the Upload link, as shown in Figure 5–52.

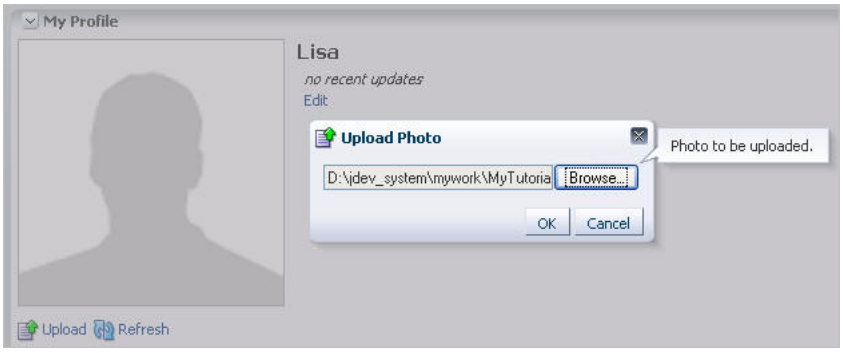
Figure 5–52 Upload Link in My Profile

4. Click **Upload**.

5. In the Upload Photo dialog, click **Browse**, then navigate to the directory containing your application images (for example, *JDEV_USER_HOME*/mywork/MyTutorialApplication/ViewController/public_html/images).

6. Locate the image lisa.PNG, then select it and click **Open**. The image path should display in the Upload Photo dialog, as shown in Figure 5–53.

Figure 5–53 Lisa.PNG in the Upload Photo Dialog



7. Click **OK**. Lisa's profile image displays in the Profile view (Figure 5–54). Notice that the image also updates in the Connections view.

Figure 5–54 Lisa's Profile Image in the Profile View

8. Try adding another user to your online network using the People Connections service. In the Connections view, under Invitations, click **Find and Invite People** (Figure 5–55).

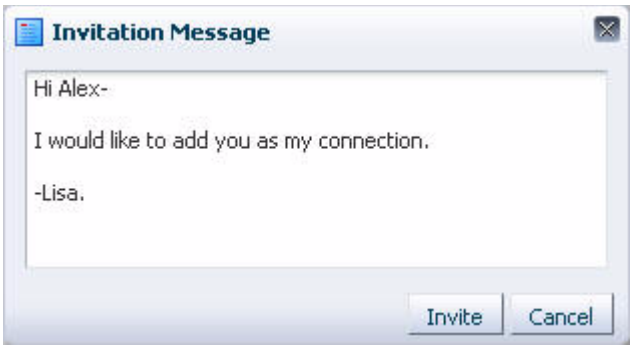
Figure 5–55 Invitations Section of the Connections View

9. In the Find User field, enter the username `Alex`, then click the arrow icon next to the field. The user Alex displays, as shown in Figure 5–56.

Figure 5–56 User Alex in Lisa's Connections View

10. Next to Alex's profile, click **Invite**.
11. In the Invitation Message dialog, personalize your invitation, as shown in Figure 5–57.

Figure 5–57 Invitation Message Dialog



12. Click **Invite**. Notice that next to Alex's profile, an invitation is now pending (Figure 5–58).

Figure 5–58 Invitation Pending in the Connections View

Also, the Sent folder under Invitations is updated. If you click **Sent(1)**, you can view and remove your invitation to Alex (Figure 5–59). Let's leave this alone for now.

Figure 5–59 Viewing an Invitation in the Connections View

13. Now, log out of the application as Lisa, then log in as Alex so that you can accept Lisa's invitation. Click **Logout** in the upper right corner of the page, then log in as the user `Alex`, using the password `welcome1`.
14. In the Connections Main View, under Invitations, notice that Alex's Received messages are updated. Click **Received(1)**, as shown in Figure 5–60.

Figure 5–60 Alex's Connection View - Received(1) Link

You should see the invitation from Lisa.

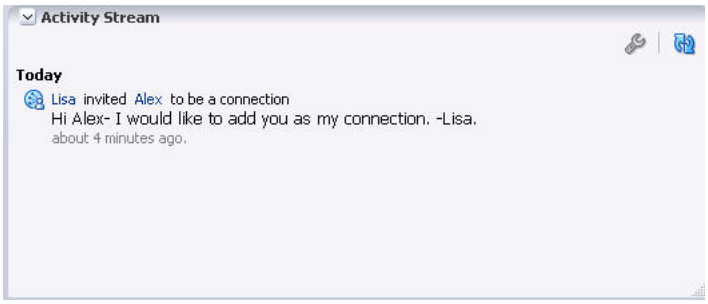
Figure 5–61 Alex's Connection View Showing Lisa's Invitation to Join Her Network

15. Click **Accept**.
16. Under Connection Lists, click **All Connections** to display your new connection with Lisa (Figure 5–62).

Figure 5–62 Alex's Connection with Lisa

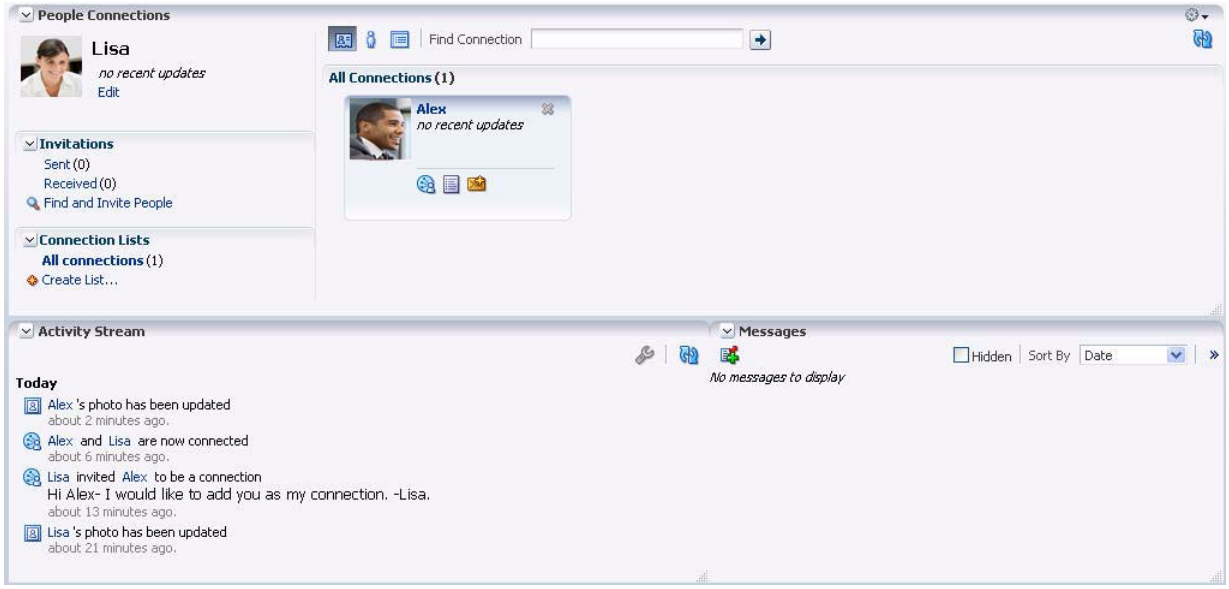
Notice now that the Activity Stream (Figure 5–63) shows Lisa's invitation to Alex.

Figure 5–63 Activity Stream



17. Optionally, you can upload Alex's image (`alex.png`) to his profile so that everyone can see it.
18. Let's log out as Alex and log back in as Lisa. Click **Logout**, then log in as `Lisa/welcome1`.
19. On Lisa's page, you should now see Alex as a connection in the Connection view, and a few updates to the Activity Stream showing the invitation, the connection, and Alex's photo update (if you performed this last step), as shown in Figure 5–64.

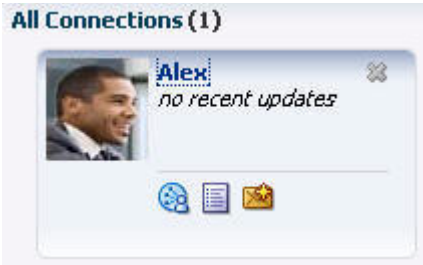
Figure 5–64 Lisa's Updated View



20. Next, check out the Message Board. The Message Board enables people connected to you (in this case, Lisa) to share information with each other. For example, as the application administrator, Lisa may want to let the Alex know about the new Message Board feature.

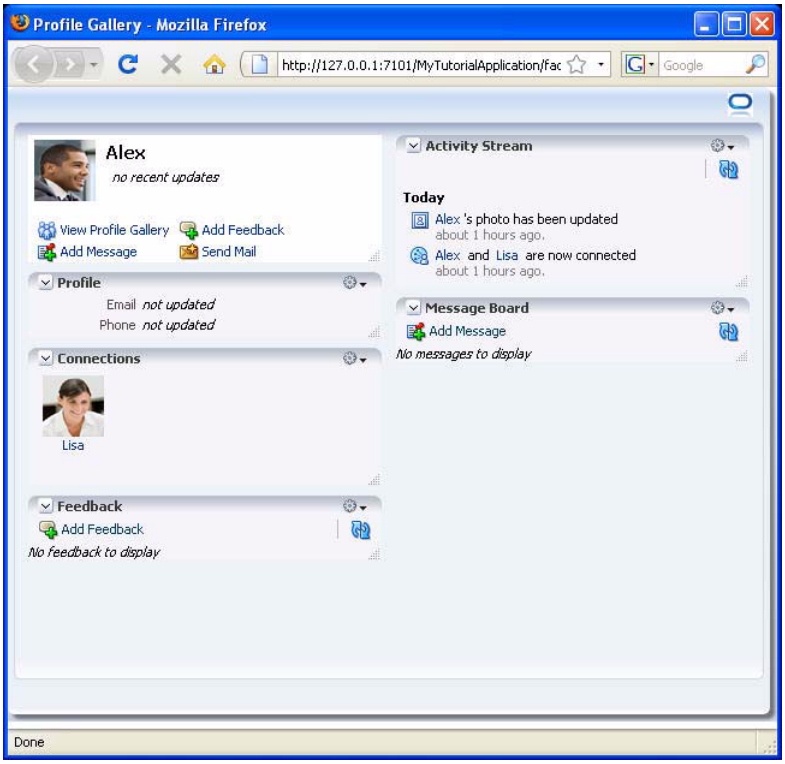
    In the Connections view, click **Alex** (Figure 5–65).

Figure 5–65 Messages View with the Add Message Icon



21. A new pop-up window displays called the Profile Gallery (Figure 5–66). Here, you can see information about Alex, such as his Activity Stream and Profile. You will also notice the Message Board. This is Alex's Message Board, where you (as Lisa) can leave messages for him.

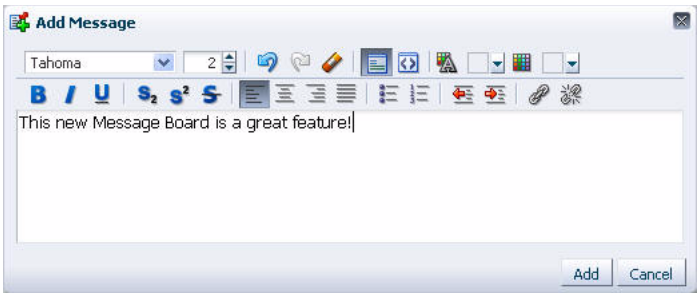
Figure 5–66 Profile Gallery



22. Under Message Board, click **Add Message**.

23. In the Add Message dialog, enter a message, such as `This new Message Board is a great feature!` Figure 5–67 shows the updated dialog.

Figure 5–67 Add Message Dialog



24. Click **Add**. The message displays on Alex's Message Board (Figure 5–68) in the Profile Gallery from Lisa's perspective.

Figure 5–68 New Message on Alex's Message Board

25. Now, let's see whether Alex can view the new message. Close the Profile Gallery window, then log out as Lisa and log back into the application as `Alex/welcome1`. You will see Lisa's message on Alex's message board.

Figure 5–69 Lisa's Message on Alex's Message Board from Alex's Perspective

These are just a few features that the People Connections service provides. For more information on using the service, see Chapter 22, "Working with the People Connections Service" in *Oracle Fusion Middleware User's Guide for Oracle WebCenter*.

Now, let's check out how to use the Links service with the application.

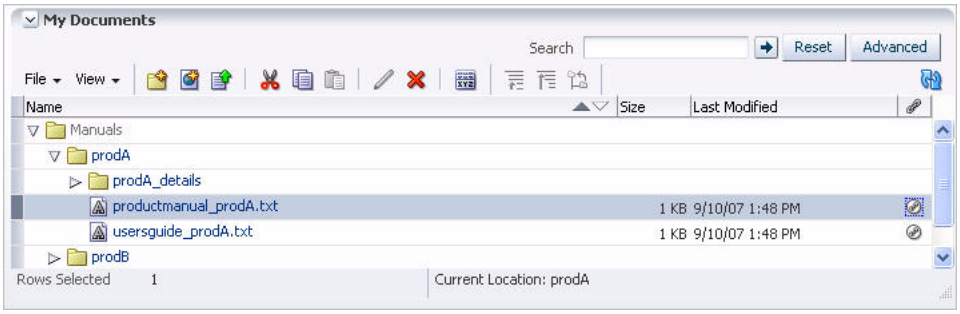
Step 10: Use the Links Service in Your Application at Runtime

The Links service provides a way to view, access, and associate related information. For example, in a list of project assignments, you can link to the specifications relevant

to each assignment. In a discussion thread about a problem with a particular task, you can link to a document that provides a detailed description of how to perform that task.

Once you connected to a database containing the WebCenter schema (in "Step 5: Create a Database Connection to the WebCenter Schema"), Oracle WebCenter Framework automatically enabled the Links service in your application. You can see the icon in the Documents - Document Manager view, for example, in Figure 5–70, to the right of a listed document.

Figure 5–70 Links Service Icon in the Documents - Document Manager View

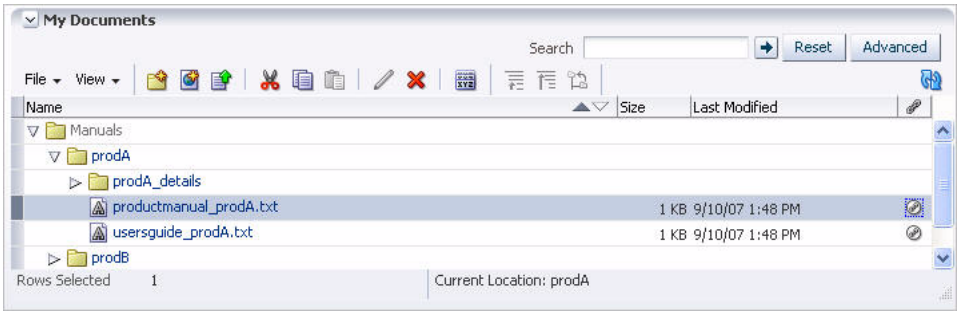


In this section, you will use the Links service to create a relationship between two documents currently listed in the Documents - Document Manager view. By following these steps, you will see how you can create relationships between objects within your application and from within your application to an external resource or URL.

To use the Links service at runtime:

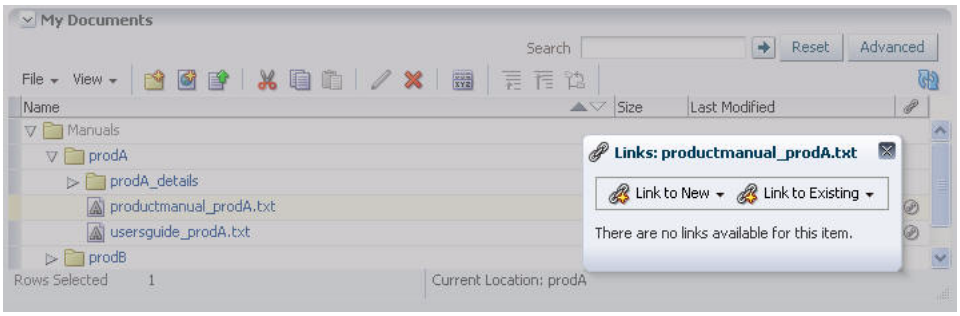
1. Ensure that **MyPage.jspx** is still running in your browser.
2. In the Documents - Document Manager view, navigate to the **Manuals** folder, then open the **prodA** folder so that you see the two files: `productmanual_prodA.txt` and `usersguide_prodA.txt` listed (Figure 5–71).

Figure 5–71 Links Service Icon in the Documents - Document Manager View



3. Let's create a relationship between the first file, `productmanual_prodA.txt`, and an external URL. To the right of the file, `productmanual_prodA.txt`, click the **Find or create Links: productmanual_prodA.txt** icon to display the Links dialog, as shown in Figure 5–72. The Links dialog displays options for linking the current document to a new document or URL or to an existing document. If links exist for the object already, they would also display in this dialog.

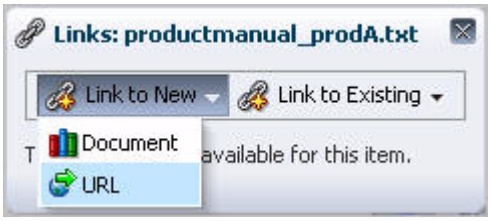
Figure 5–72 Links Dialog



4. Since the file we want to *link to* already exists, click **Link to New**, then click **URL** (Figure 5–73). If you choose to link to a new *document*, a dialog displays where you can upload a new document to the content repository and link to it.

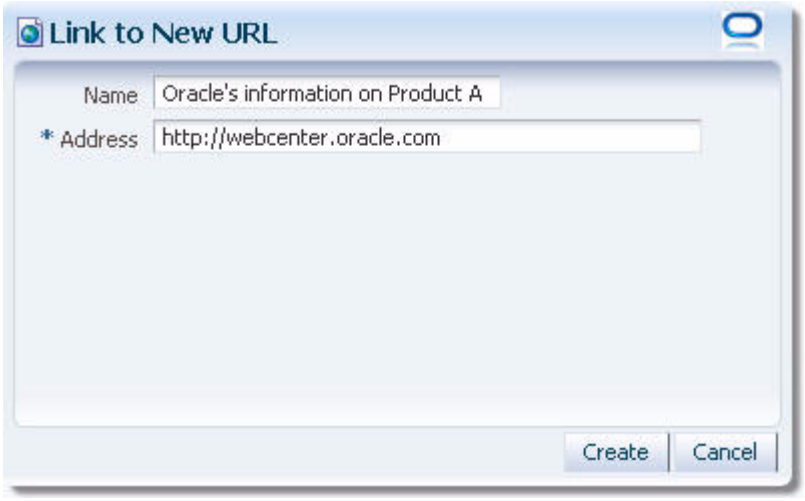
   Notice that you can also choose **Link to Existing** and select a document that already exists within the application.

Figure 5–73 Link to New URL Menu Option



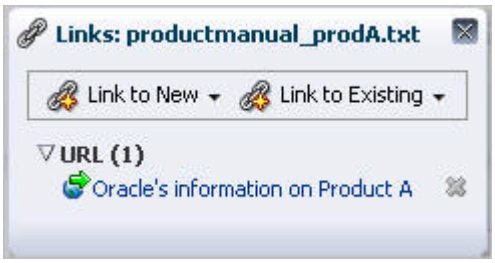
5. In the Link to New URL dialog, in the Name field, enter `Oracle's information on Product A`, then enter `http://webcenter.oracle.com` for the address (Figure 5–74).

Figure 5–74 Linking to a New URL



6. Click **Create**. The new link displays in the Links dialog, as shown in Figure 5–75.

Figure 5–75 New URL in the Links Dialog



7. Click the new link to view the related URL Now, if you or other users wants to find related information on Product A, you can click the icon to view the associated links.

For more information about the Links service, see Chapter 22, "Integrating the Links Service" in *Oracle Fusion Middleware Developer's Guide for Oracle WebCenter* and Chapter 18, "Working with the Links Service" in *Oracle Fusion Middleware User's Guide for Oracle WebCenter*.

Step 11: Use the Mail Service with Your Application (Optional)

If you have access to an email server (such as the one you use with your own personal email), you can easily configure your application to use the Mail service. The Mail service enables you to read, send, and organize your email directly within a custom WebCenter application. You can find detailed instructions for using the Mail service in the *Oracle Fusion Middleware Developer's Guide for Oracle WebCenter*.

To add the Mail service to your application:

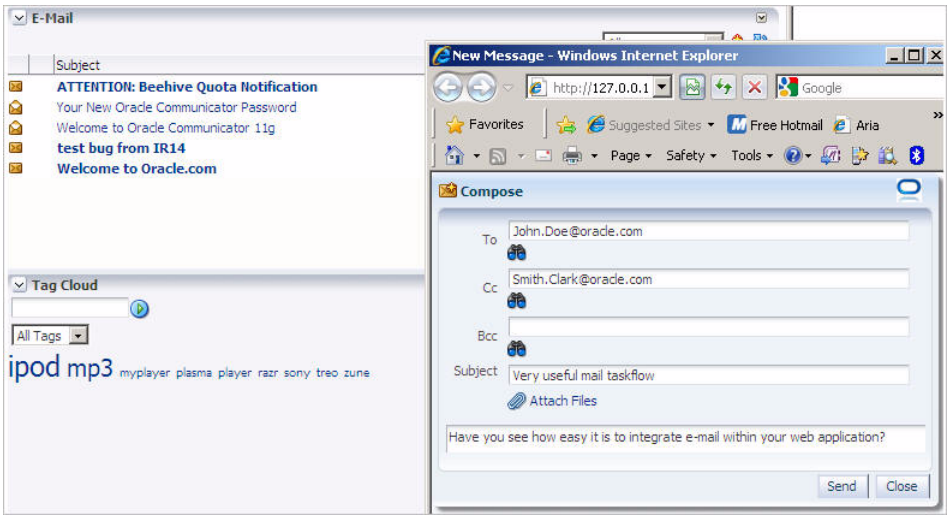
1. Follow the steps in "Step 2: Create Users and Roles for the Application" to create a user that exists on your mail server (for example, your own user name and password). The application must have a user associated with it that also exists on the mail server.
2. Follow the instructions in Chapter 17, "Integrating the Mail Service" in the *Oracle Fusion Middleware Developer's Guide for Oracle WebCenter* to create a connection to your mail server and add the Mail task flow to your application.
3. Run `MyPage` to your browser and log in with your user name and password.

   You can click the dropdown list in the upper right corner to sort the view of your inbox by time range, as shown in Figure 5–76.

Figure 5–76 Mail Task Flow at Runtime

You can also click the plus sign to display the New Message dialog, as shown in Figure 5–77.

Figure 5–77 Mail Task Flow with New Message Dialog at Runtime



For more information on using the Mail service, see Chapter 20, "Working with the Mail Service" in *Oracle Fusion Middleware User's Guide for Oracle WebCenter*.

You can learn more about adding the Search, Documents, Tags, Links, People Connections, and Mail services to a custom WebCenter application in the *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 you have added a few services to our application, you can learn how to build and add portlets to your application in Chapter 6, "Building Portlets and Wiring Them in Your Application."

6

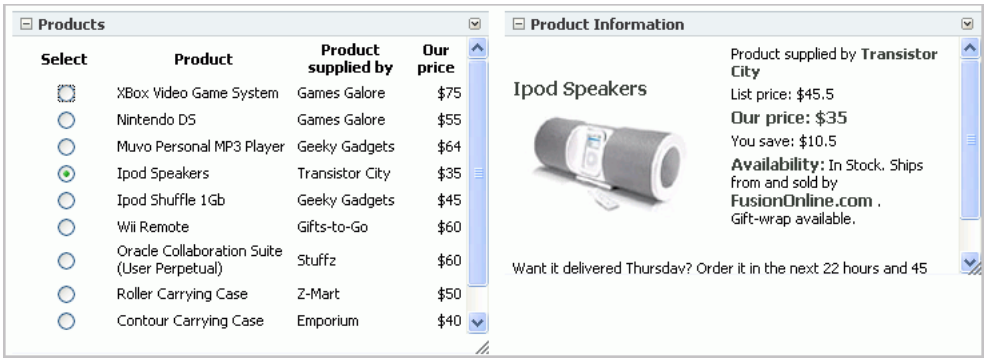
Building Portlets and Wiring Them in Your Application

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 will 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 6–1 shows how the portlets section of your page will look.

Figure 6–1 Portlets Section of MyPage.jspx



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 5, "Adding Oracle WebCenter 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" 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.

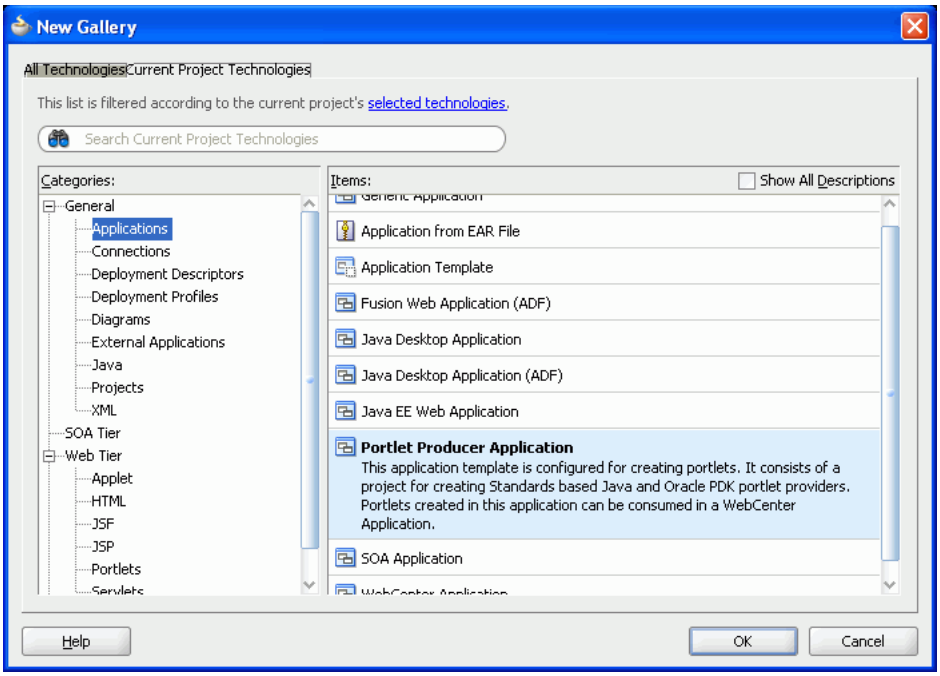
Note: The steps to build this portlet rely on the naming convention we've used, so follow the steps carefully. If you do not use the names we've provided, you may not achieve the same results.

In this step, you will create an application based on the Portlet Producer template, then build a standards-based portlet. Afterward, you will consume the portlet into our Tutorial application. Figure 6–2 shows the portlet at runtime in `MyTutorialApplication`.

Figure 6–2 Standards-Based Portlet (JSR 168) at Runtime in MyTutorialApplication

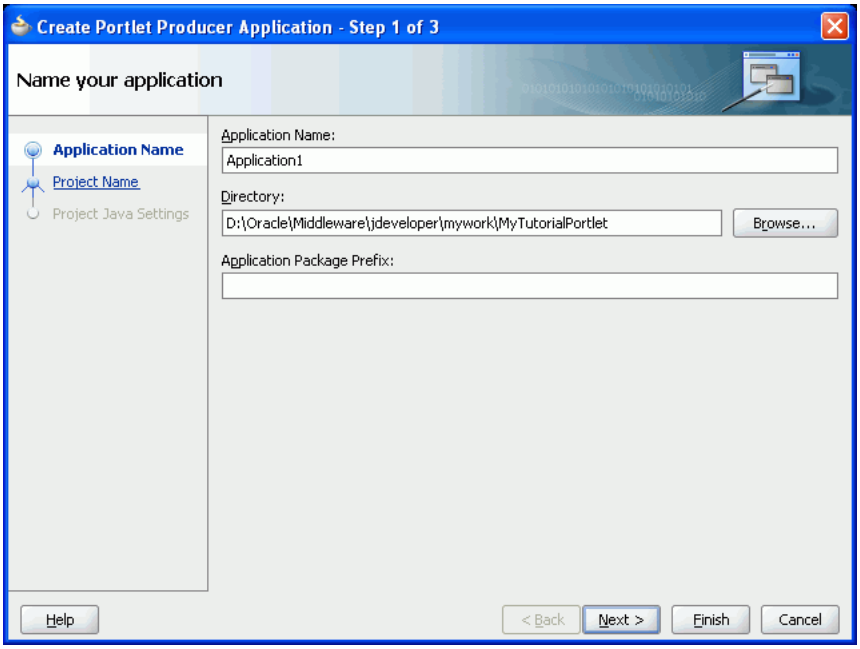
1. In Oracle JDeveloper, create a new application based on the **Portlet Producer Application** template (Figure 6–3), then click **OK**. To do so, choose **New** from the File menu to display the New Gallery.
2. In the New Gallery, choose Applications from the Categories list, then choose **Portlet Producer Application**.

Figure 6–3 Creating a New Portlet Producer Application



3. In the Create Portlet Producer Application wizard, in the Application Name field, enter `MyTutorialPortlet`.
4. By default, the Directory field should contain the directory where the application will reside (Figure 6–4). You can change the directory location, if you like, but let's leave it as it is for the purposes of the Tutorial.

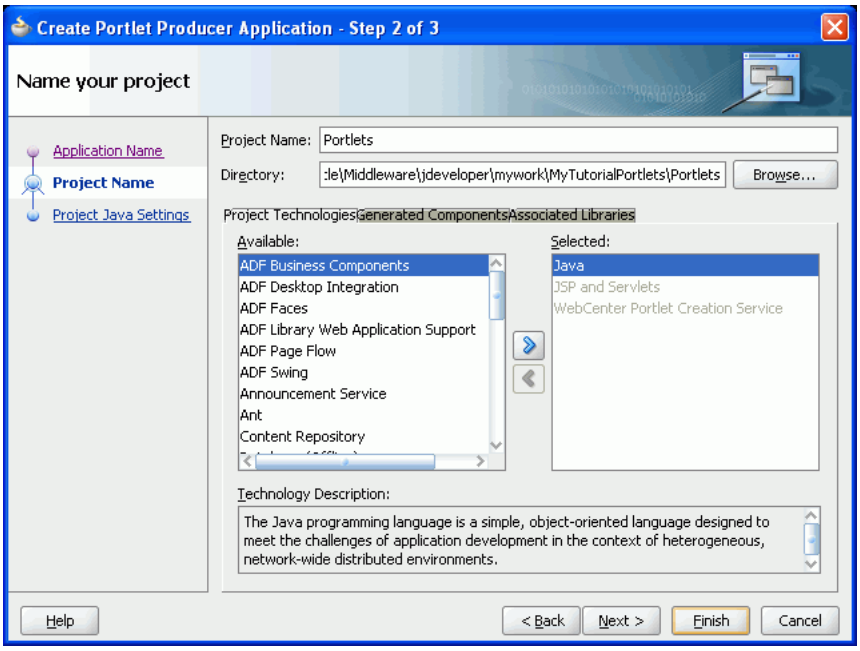
Figure 6–4 Creating a Portlet Producer Application - Step 1



5. Click **Next**.

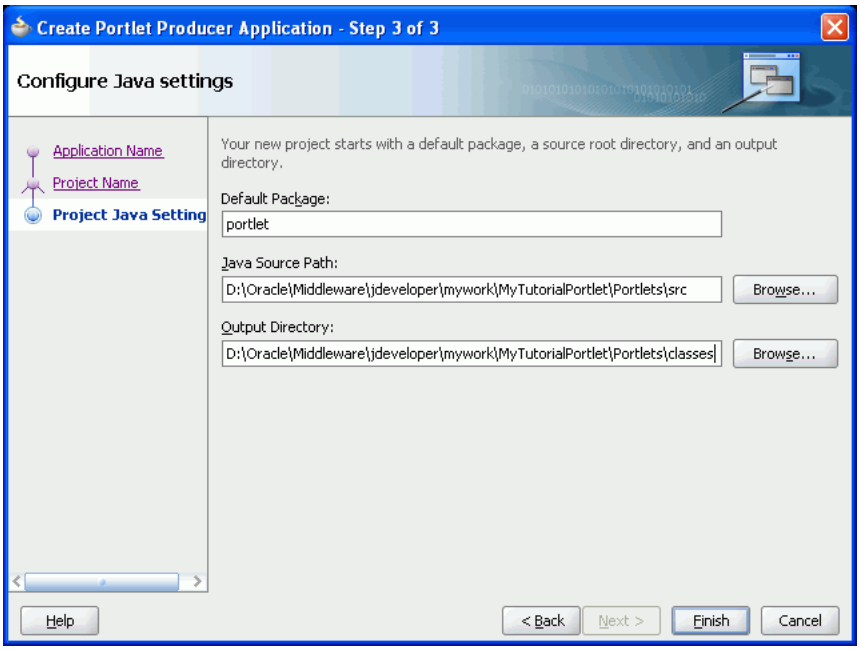
6. On the Project Name page of the wizard (Figure 6–5), let's leave the default name: `Portlets`, and click **Next**.

Figure 6–5 Creating a Portlet Producer Application - Step 2



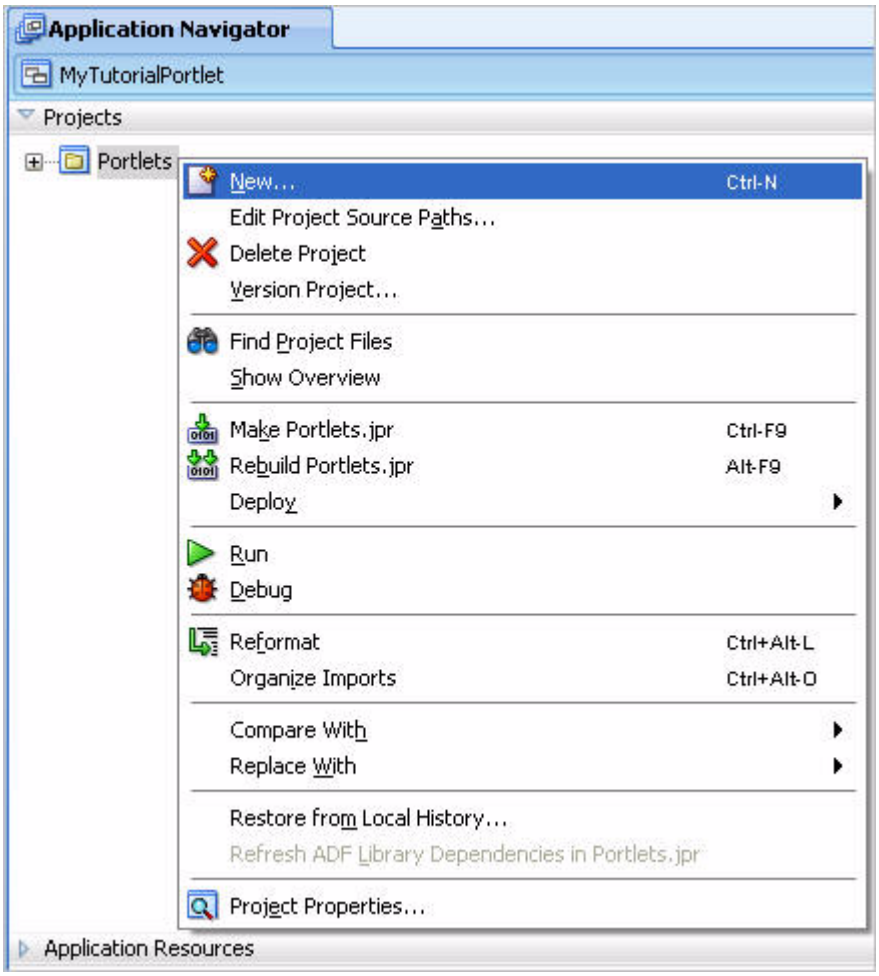
7. On the last page of the wizard, you can configure the Java settings. You can see that the default package contained in the portlet producer application is `portlet` (Figure 6–6). For the purposes of this Tutorial, let's leave the default options and click **Finish**.

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



8. In the Application Navigator, under your new `MyTutorialPortlet` application, right-click **Portlets**, and select **New** (Figure 6–7).

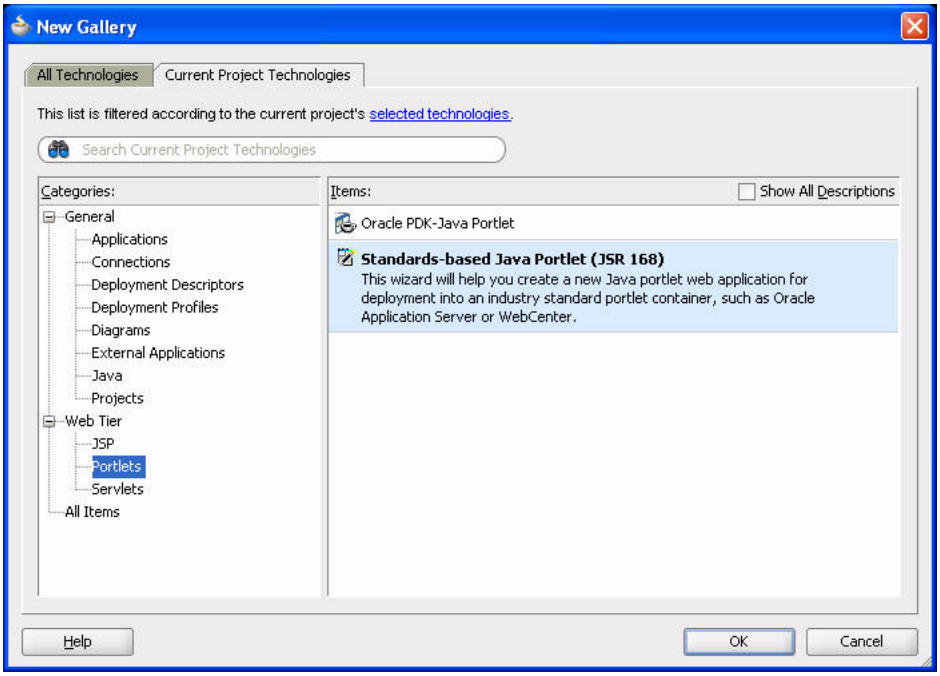
Figure 6–7 Selecting New from the Context Menu



9. Click the **Current Project Technologies** tab.

10. In the Categories list as shown in Figure 6–8, expand the **Web Tier** category, and select **Portlets**.

Figure 6–8 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.

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

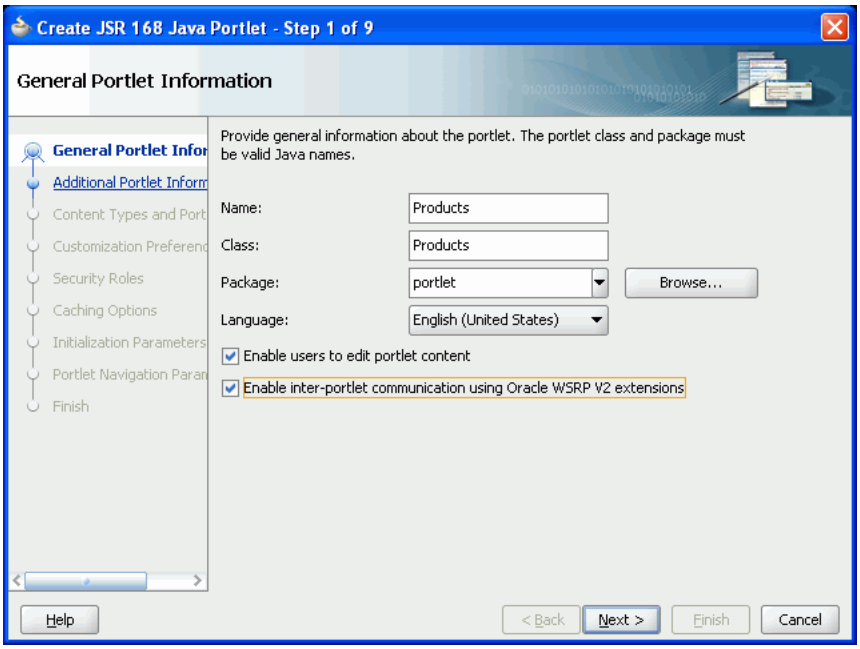
    The JSR 168 Java Portlet wizard displays, which generates a skeleton for the portlet. We will later add our own logic to the portlet. Let's see how this is done.

12. 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.

13. 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 will need these parameters later on, when we enable the OmniPortlet and this portlet to communicate with each other. Figure 6–9 shows how the General Portlet Information page should now look.

Figure 6–9 Creating a JSR 168 Java Portlet - General Portlet Information



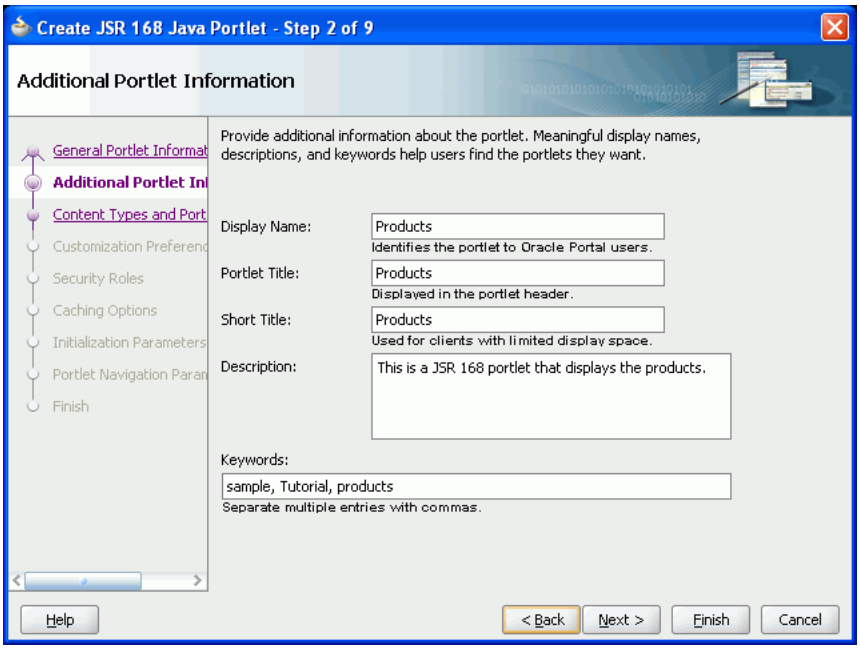
14. Click **Next**.

15. 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 6–1. Figure 6–10 shows the resulting Additional Portlet Information page.

Table 6–1 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 10*g* 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 6–10 Creating a JSR 168 Java Portlet - Additional Portlet Information

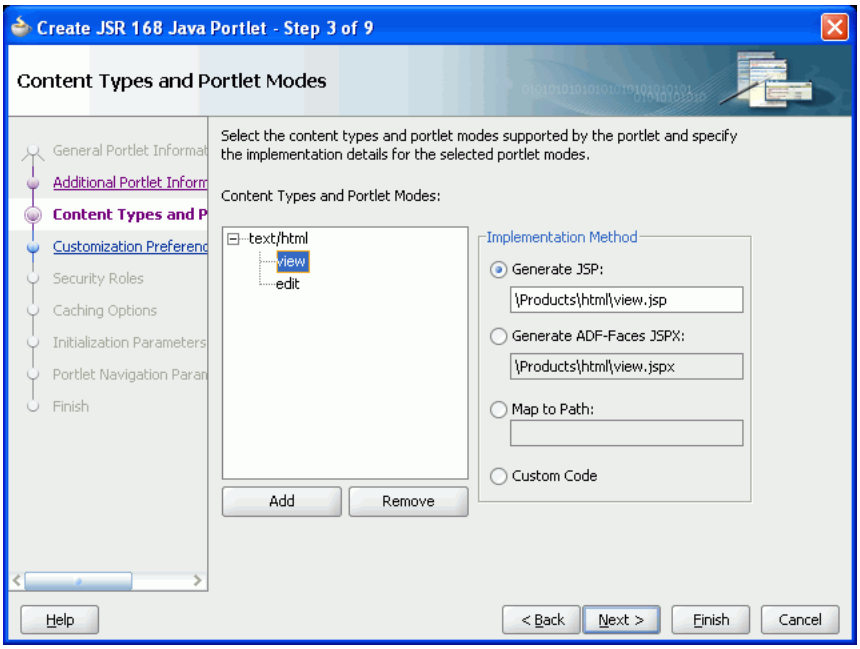


16. Click **Next**.

17. 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 6–11. These controls enable you to specify whether you want to generate JSP for the portlet, or use your own custom JSP code.

Figure 6–11 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.

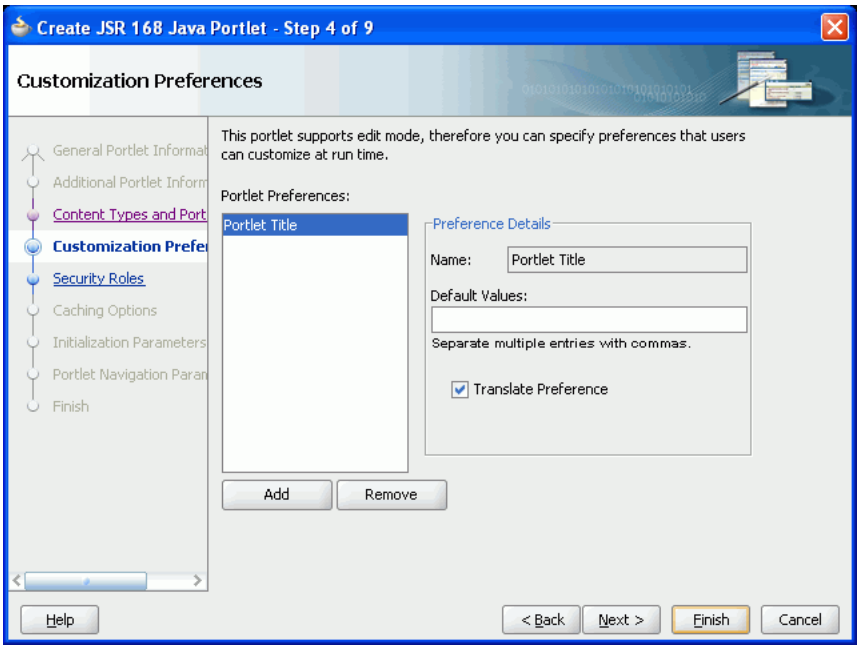
18. Click **Next**.

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

19. On the Customization Preferences page, let's leave the default values and click **Next** (Figure 6–12).

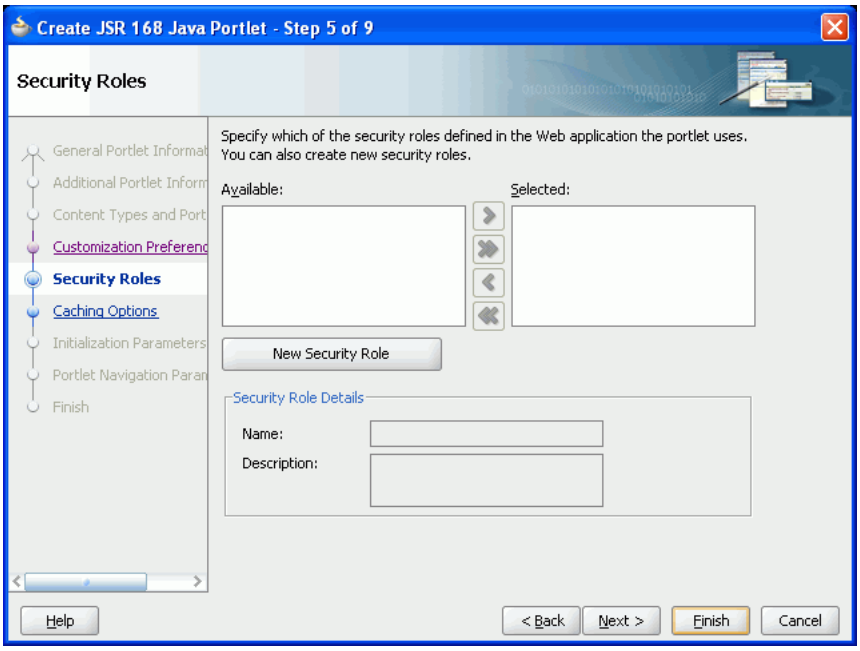
    Although we're not going to do anything with this page now, in the future you can use it to add other customization options for the portlet. For example, if your portlet accepted a `Zip Code` parameter, you might want to allow users to personalize the Zip Code label. If this were the case, you would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

Figure 6–12 Creating a JSR 168 Java Portlet - Customization Preferences



20. On the Security Roles page, click **Next** (Figure 6–13). The Security Roles page is used to specify which of the application's security roles you want to establish for this portlet.

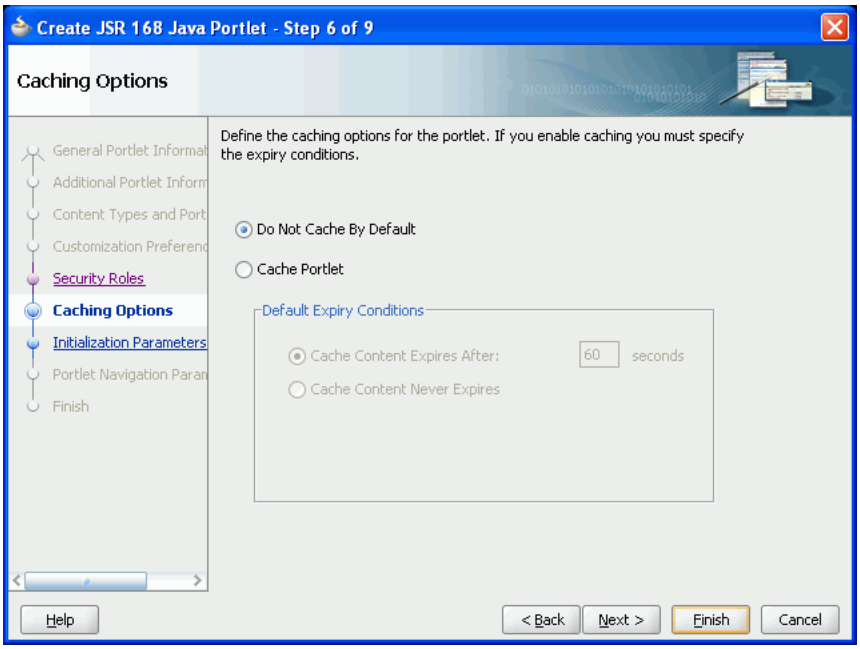
Figure 6–13 Creating a JSR 168 Java Portlet - Security Roles



21. On the Caching Options page, leave the default option and click **Next** (Figure 6–14).

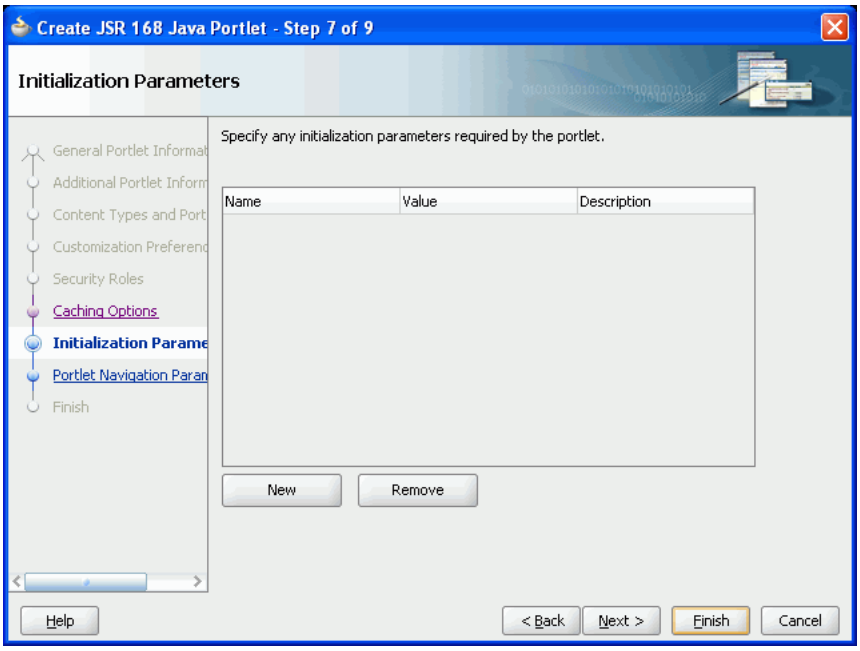
    The settings on this page enable you to define expiry-based caching for your portlet. You do not need any caching conditions now.

Figure 6–14 Creating a JSR 168 Java Portlet - Caching Options



22. On the Initialization Parameters page, click **Next** (Figure 6–15). Our portlet does not require any initialization parameters.

Figure 6–15 Creating a JSR 168 Java Portlet - Initialization Parameters



23. 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.

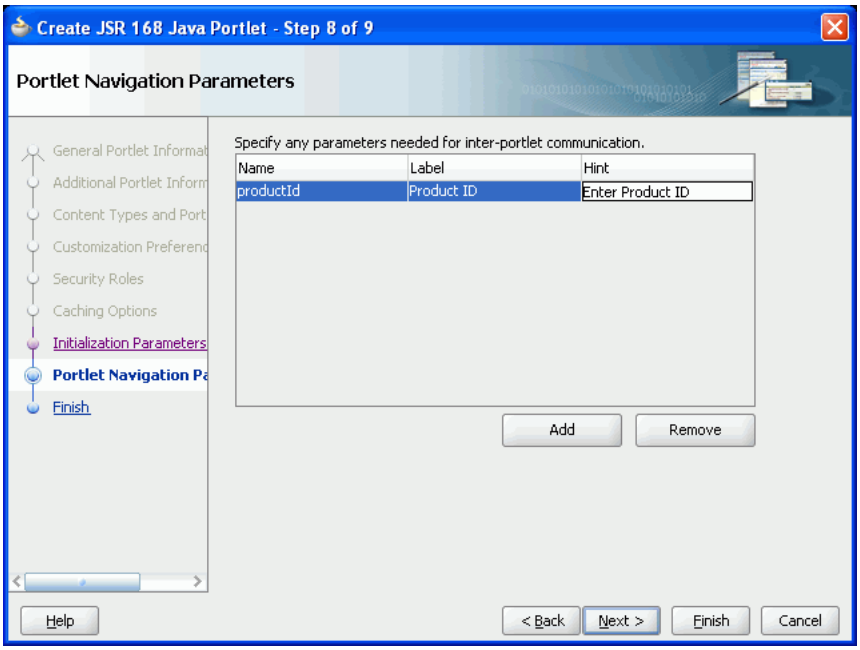
Click **Add**.

24. 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 6–16 shows the updated Portlet Navigation Parameters page.

Figure 6–16 Creating a JSR 168 Java Portlet - Portlet Navigation Parameters

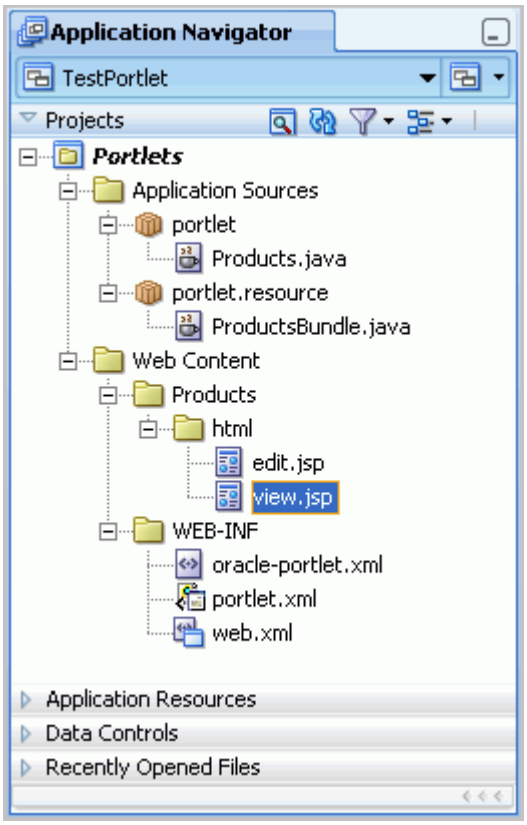


25. Click **Next**.

26. 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 6–17.

Figure 6–17 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.
  - **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.

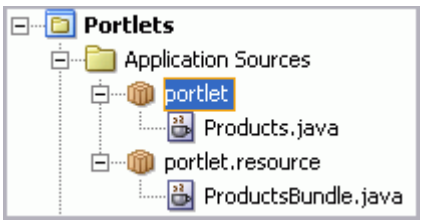
27. Save all your files. In the next step, we will create a JavaBean to store all the portlet information you just generated.

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.

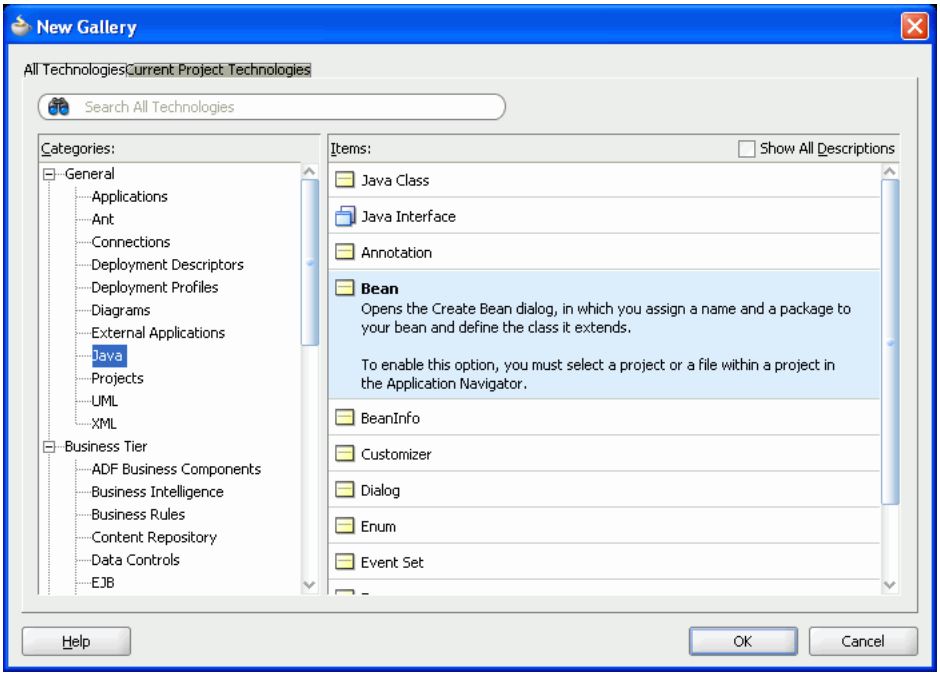
1. In the Application Navigator, right-click the **portlet** package, and choose **New**. Figure 6–18 shows the `portlet` package in the Application Navigator.

Figure 6–18 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 6–19).

Figure 6–19 Choosing the JavaBean in the New Gallery



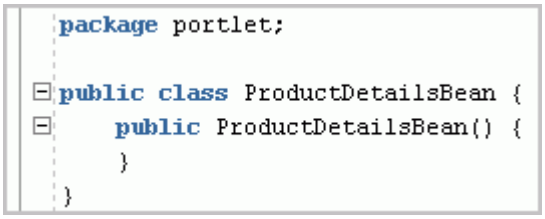
4. In the Create Bean dialog that displays, enter the following information to set up the new JavaBean (Figure 6–20):
   - Name: `ProductDetailsBean`
   - Package: `portlet`
   - Extends: `java.lang.Object`

Figure 6–20 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 6–21).

Figure 6–21 ProductDetails JavaBean in the Design View

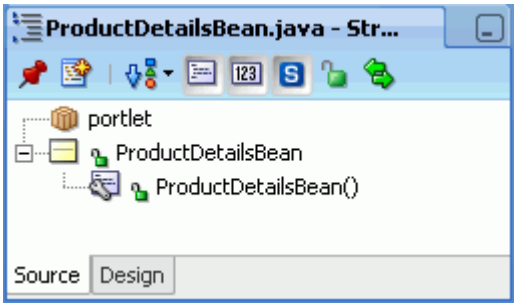


```
package portlet;

public class ProductDetailsBean {
    public ProductDetailsBean() {
    }
}
```

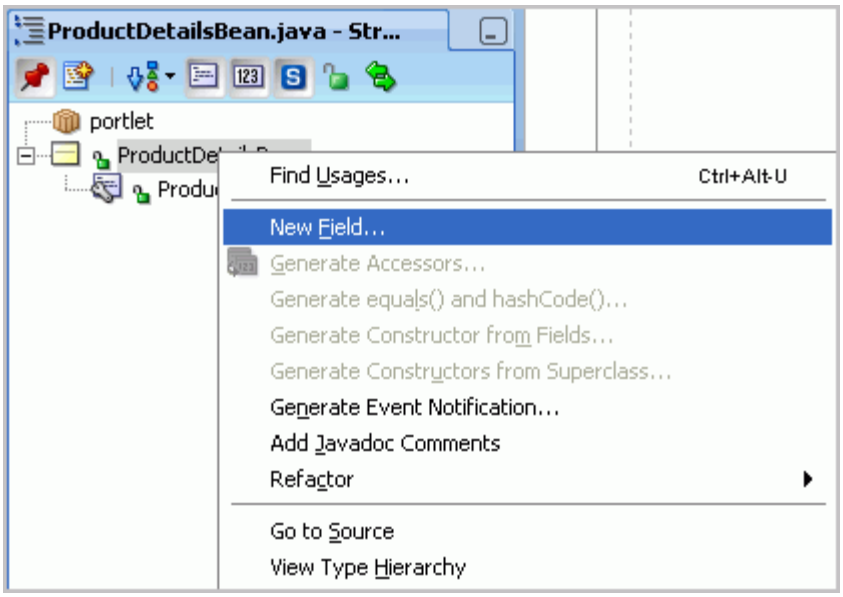
Figure 6–22 shows the JavaBean in the Structure window. As previously mentioned in Chapter 3, "Creating a WebCenter Application with a Customizable Page," you can use the pushpin in the Structure window to freeze and unfreeze the current view. Ensure that you have selected the `ProductDetails Bean` in the Design view, then toggle the pushpin so that you see the `ProductDetails Bean` in the Structure window.

Figure 6–22 ProductDetailsBean in the Structure Window



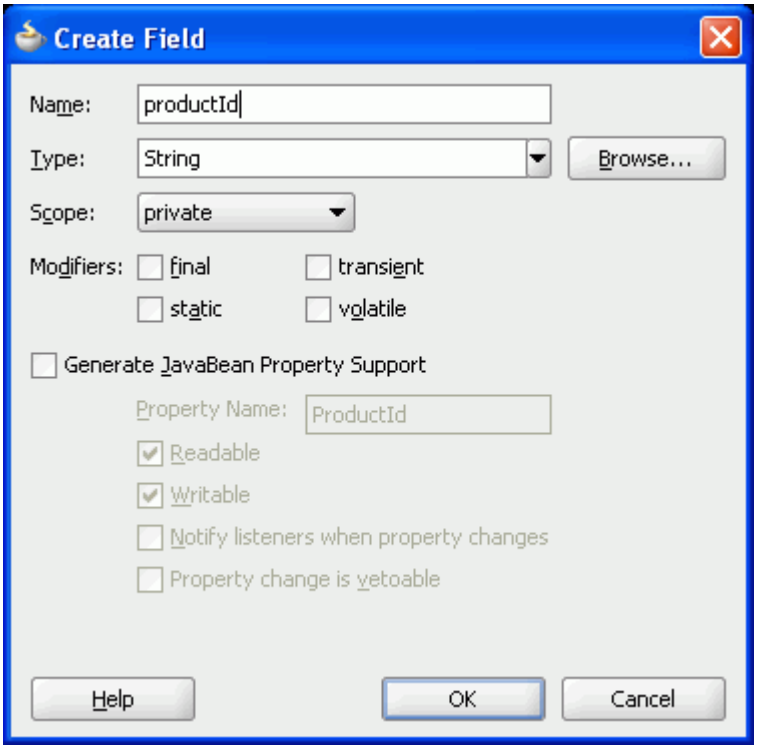
6. Now that we've set up our JavaBean, let's add the information we need for the portlet. In the Structure window, right-click the **ProductDetailsBean**, then choose **New Field** from the context menu (Figure 6–23).

Figure 6–23 Creating a New Field for the ProductDetails Bean



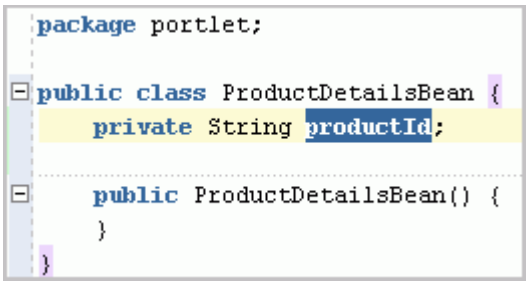
7. In the Create Field dialog, in the **Name** field, enter `productId`. This name represents the name of the product in our database schema.
8. Ensure the Type is set to `String`.
9. Ensure the Scope is set to **private** (Figure 6–24).

Figure 6–24 Create Field Dialog



10. Click **OK**. The new field displays in the Design view of the JavaBean, as shown in Figure 6–25.

Figure 6–25 New Field in the 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 6–26).

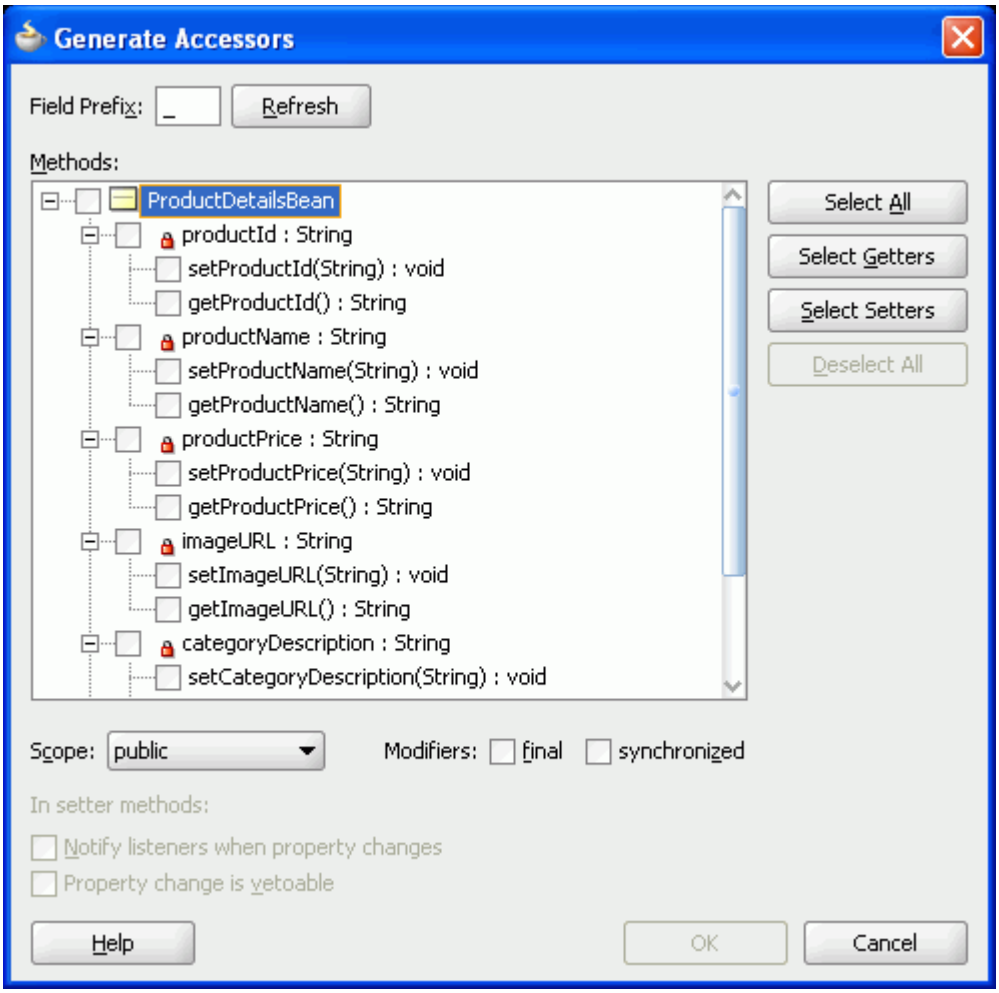
Figure 6–26 ProductsBean with the Six Fields

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.

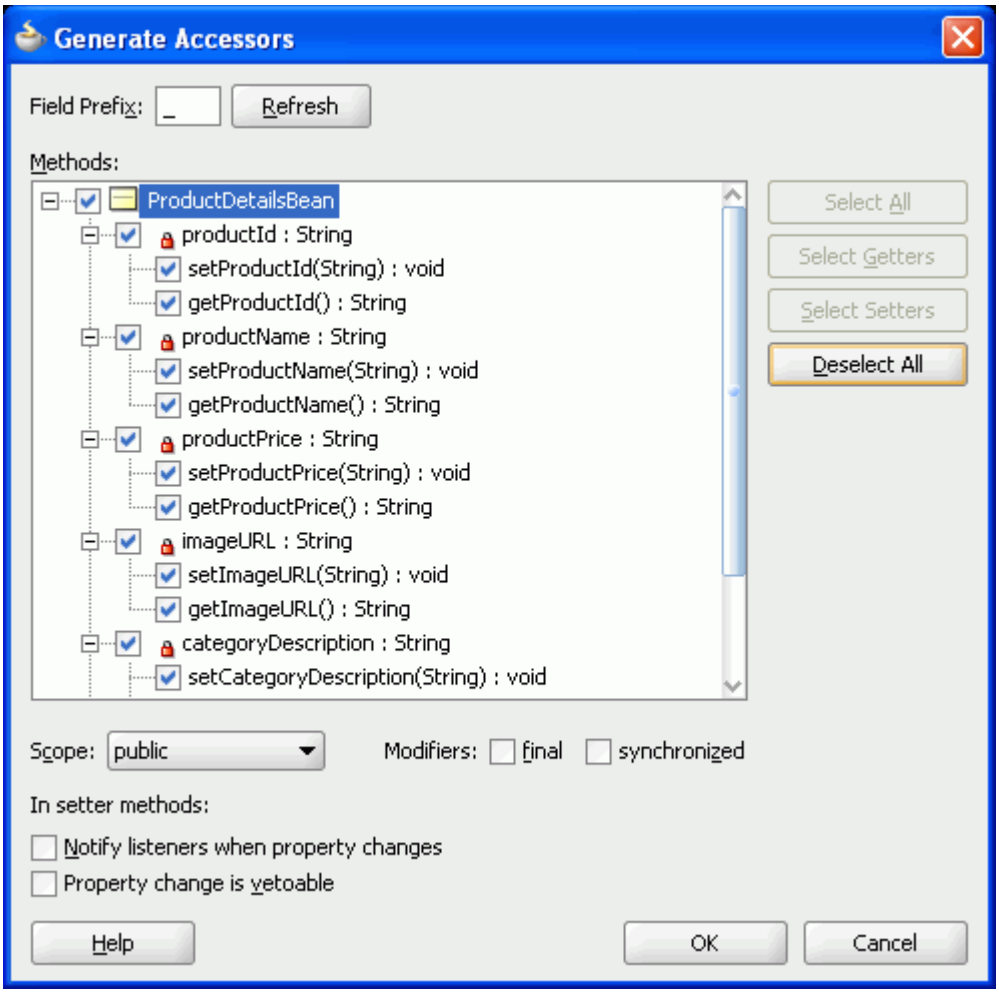
    The Generate Accessors dialog displays (Figure 6–27).

Figure 6–27 Generate Accessors Dialog



13. Click **Select All** to select all the fields you created for this bean (Figure 6–28).

Figure 6–28 Generate Accessors Dlalog with all Fields Selected



14. Click **OK** to generate the accessors for these fields. Oracle JDeveloper generates the code as shown in Example 6–1:

Example 6–1 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.

16. In the New Gallery, click the **Java** category, then click the **Bean** item to display the Create Bean dialog.

17. In the Name field, enter `ProductsBean` and `java.lang.Object` in the Extends field, as shown in Figure 6–29, then click **OK**.

Figure 6–29 Creating the ProductsBean

18. Replace the code of `ProductsBean.java` in the Source view with the code in Example 6–2.

> **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 6–2 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 6–30.

Figure 6–30 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;
    }
}
```

19. Save all your files.

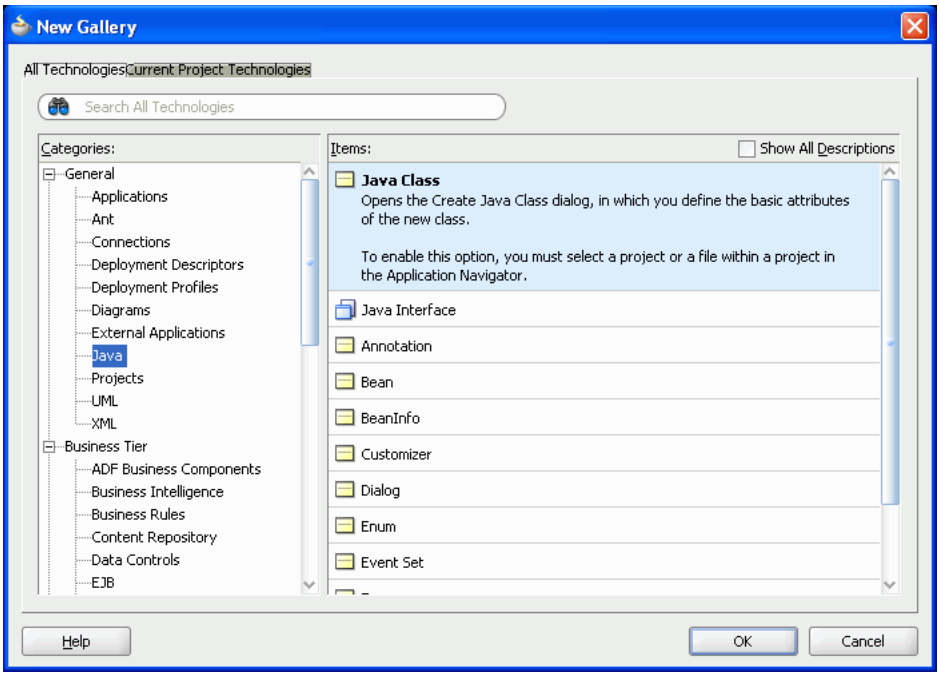
In the next steps, we'll set up the connection between our JavaBeans and the database schema that contains the data we want to show in our portlet.

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. The database schema you connect to is the one you established in Chapter 2, "Preparing for the Tutorial."

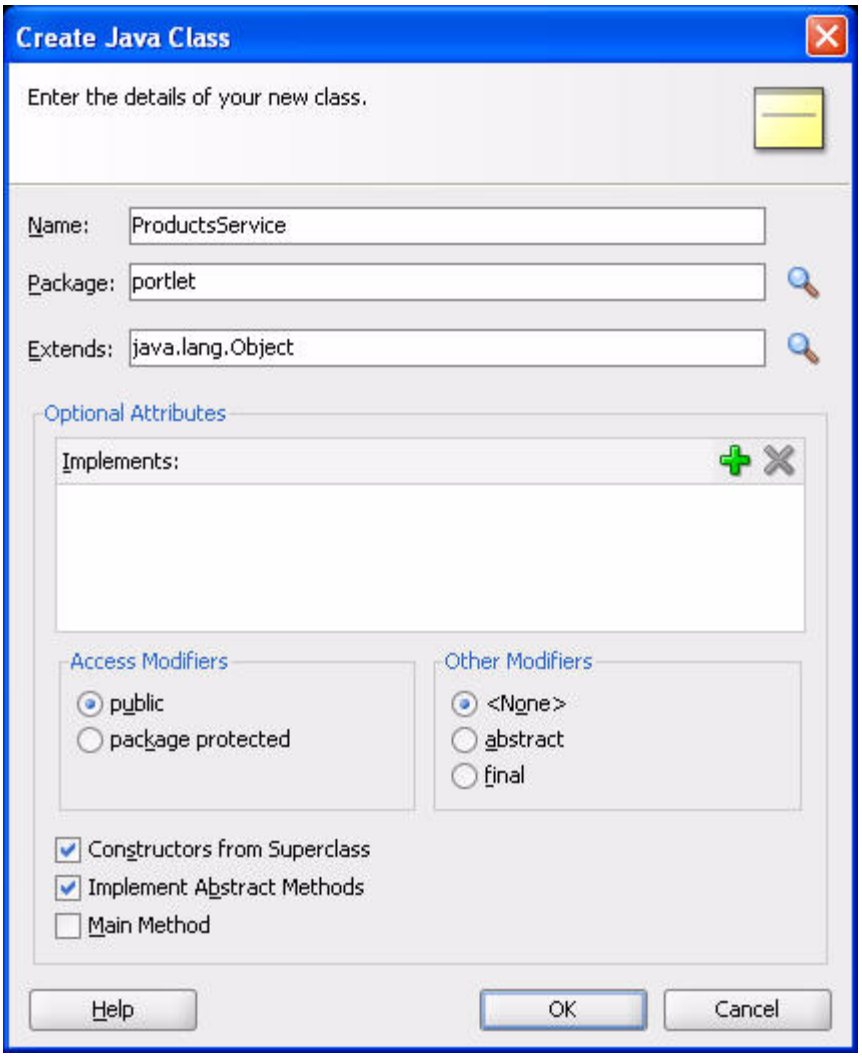
1. In the Application Navigator, right-click the **portlet** package, then choose **New** from the context menu.
2. In the New Gallery, click **Java**, then choose **Java Class** from the Items list (Figure 6–31).

Figure 6–31 Choosing the Java Class Option in the New Gallery



3. Click **OK**.
4. In the Create Java Class dialog, in the Name field, enter the name: `ProductsService`, as shown in Figure 6–32.

Figure 6–32 Create Java Class Dialog



5. Leave the rest of the default values in this dialog, as we will overwrite them in the next step, then click **OK**.
6. In the `ProductsService.java` file that displays, replace all the automatically-generated code with the code shown in Example 6–3. 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 (`fod`) schema

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

Example 6–3 ProductsService.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 6–16.

   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 6–4 shows the section of the code in the `Products.java` file where the method is located.

Example 6–4 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 6–4 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 6–4 shows the updated section.

Example 6–5 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 intepreted by the
            // container as navigational parameters as the names match the names
of
            // the declared parameters.
            response.setRenderParameter(PARAMETER1, param1);
        } else {
            // Determine which action.
```

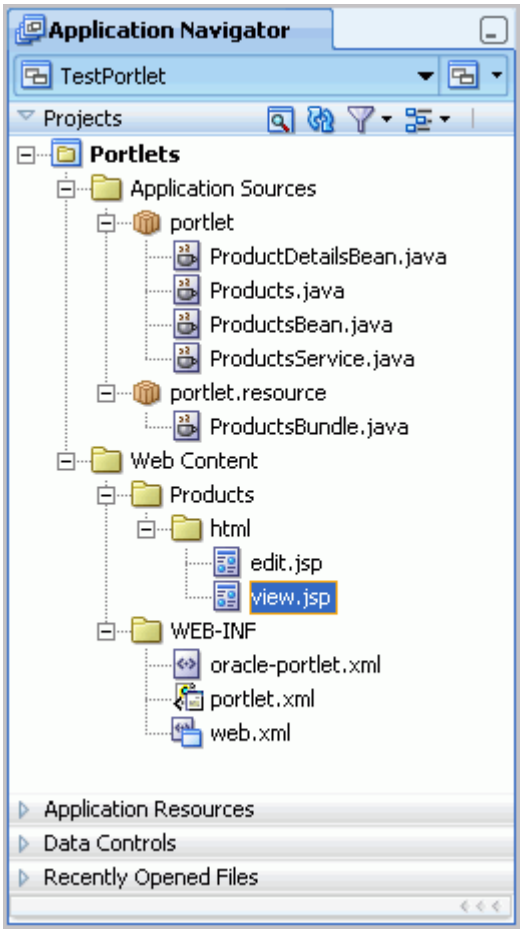
```
                String okAction = request.getParameter(OK_ACTION);
                String applyAction = request.getParameter(APPLY_ACTION);

                if (okAction != null || applyAction != null) {
                    // Save the preferences.
                    PortletPreferences prefs = request.getPreferences();
                    String param = request.getParameter(PORTLETTITLE_KEY);
                    prefs.setValues(PORTLETTITLE_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,** and **html**, pen the `view.jsp` file (Figure 6–33).

Figure 6–33 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 6–6 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 6–6 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"> </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`.

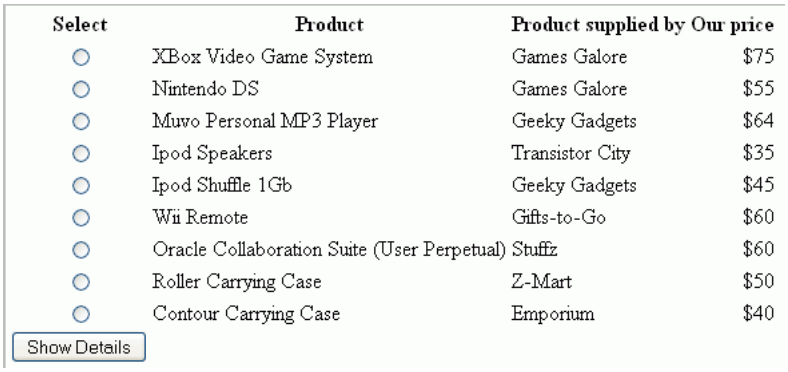
Now that you have established the connection to the database and set up the portlet to use the new JavaBean and Java Class to get the appropriate product information for the portlet, you are ready to include the portlet in a WAR file. WAR stands for *web archive*, and it packages all the resources, portlets,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

Step 4: Test and Deploy the Standards-Based Portlet

In this lesson, you will learn how to deploy the standards-based portlet to your local 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 Integrated WebLogic Server by choosing **Start Server Instance** from the **Run** menu.
2. Before we deploy the portlet, let's quickly compile and test it. In the Application Navigator, under Web Content then **Products**, right-click **view.jsp** and choose **Run**. The portlet should compile and display in your browser, as shown in Figure 6–34.

Figure 6–34 Testing the Standards-Based Portlet

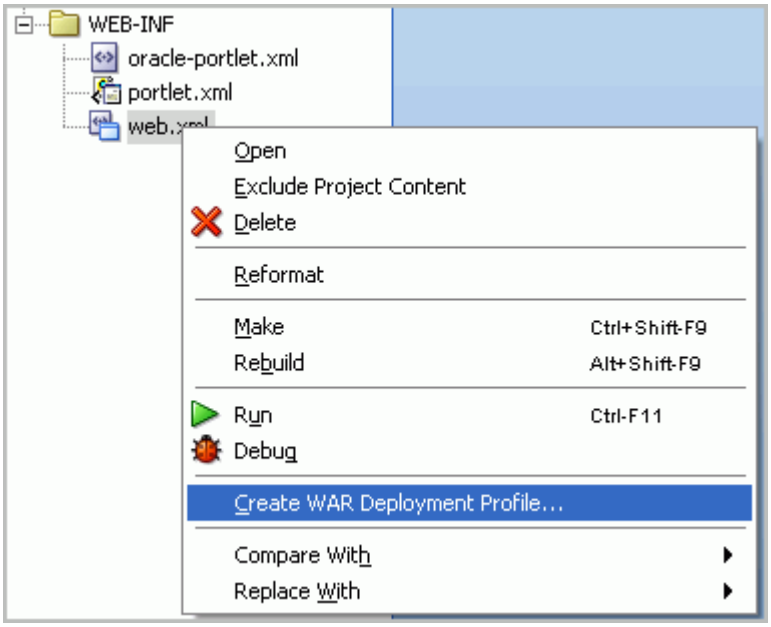


Select	Product	Product supplied by	Our price
○	XBox Video Game System	Games Galore	$75
○	Nintendo DS	Games Galore	$55
○	Muvo Personal MP3 Player	Geeky Gadgets	$64
○	Ipod Speakers	Transistor City	$35
○	Ipod Shuffle 1Gb	Geeky Gadgets	$45
○	Wii Remote	Gifts-to-Go	$60
○	Oracle Collaboration Suite (User Perpetual)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

Show Details

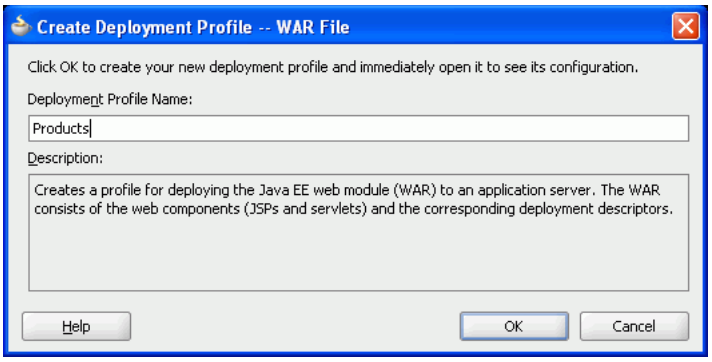
3. In the Application Navigator, under **Portlets**, **Web Content**, **WEB-INF**, right-click the `web.xml` file, then choose **Create WAR Deployment Profile** from the context menu (Figure 6–36).

Figure 6–35 Create WAR Deployment Profile Menu Option



4. In the Create Deployment Profile dialog, name the Deployment Profile `Products`, then click **OK** (Figure 6–36).

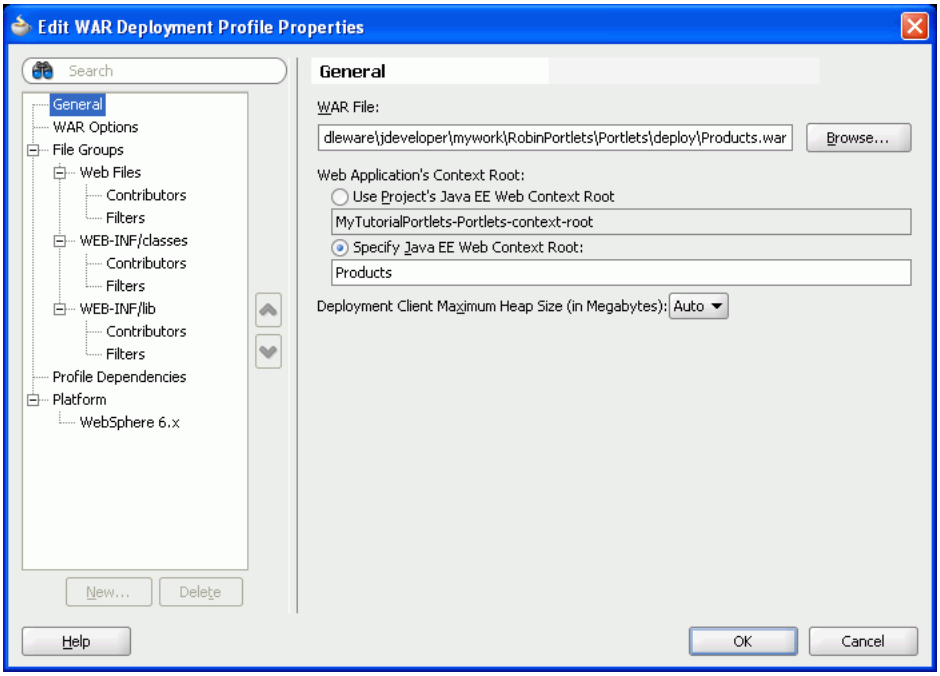
Figure 6–36 Create Deployment Profile - WAR File Dialog



5. On the General tab of the WAR Deployment Profile Properties dialog, 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 6–37.

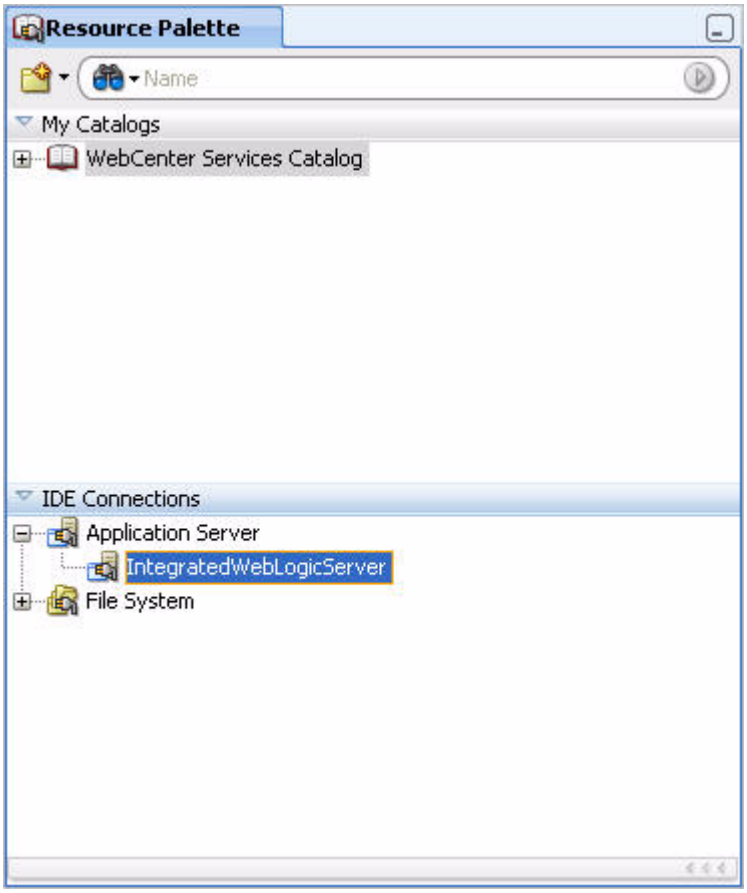
Figure 6–37 WAR Deployment Profile Properties -- Setting the Context Root



6. Click **OK** to finish updating the properties, then click **OK** in the Project Properties dialog to finish creating the WAR deployment profile.

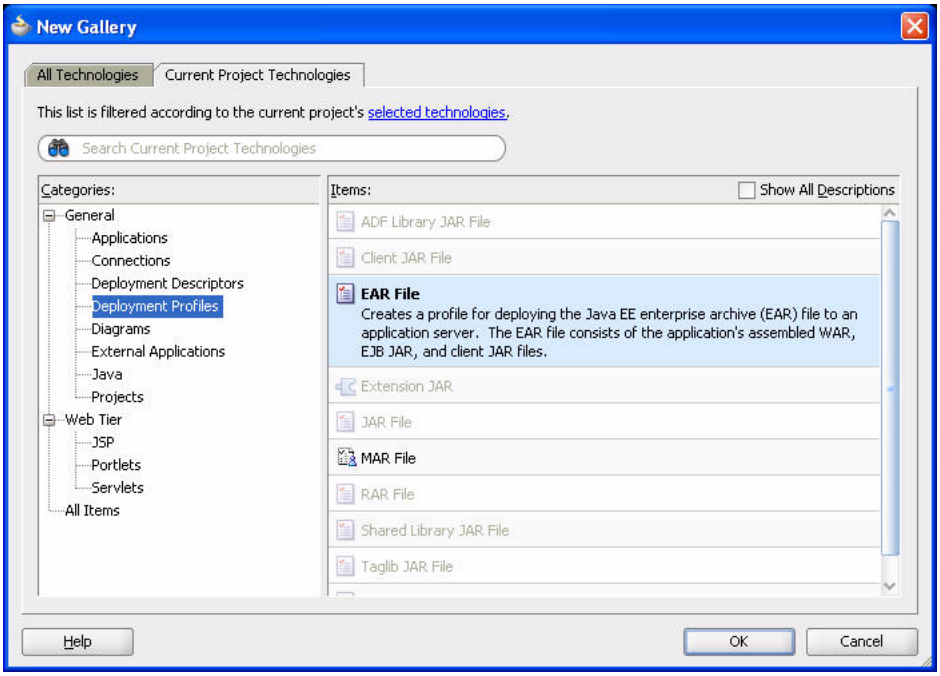
7. Oracle WebCenter includes its own default connection to the Integrated WLS Server. You can see the connection, called **IntegratedWLSConnection**, in the Resource Palette, under **IDE Connections** (Figure 6–38).

Figure 6–38 IntegratedWebLogicServer in the Resource Palette



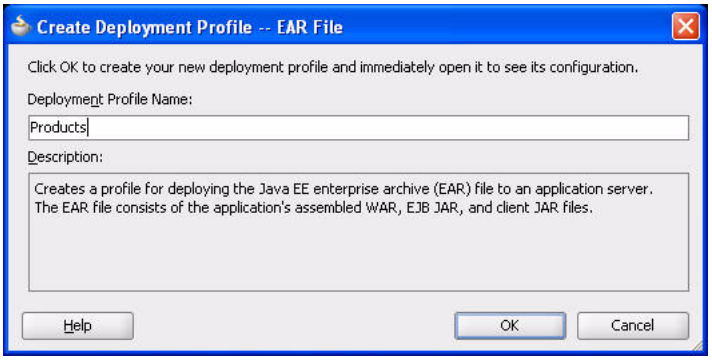
8. Now, create an EAR file for the Deployment Profile. From the Application menu, choose **Deploy**, then **New Deployment Profile**.
9. In the New Gallery, ensure **Deployment Profiles** is selected, then choose **EAR** File from the Items list (Figure 6–39).

Figure 6–39 Creating an EAR File for the Deployment Profile of the Products Portlet



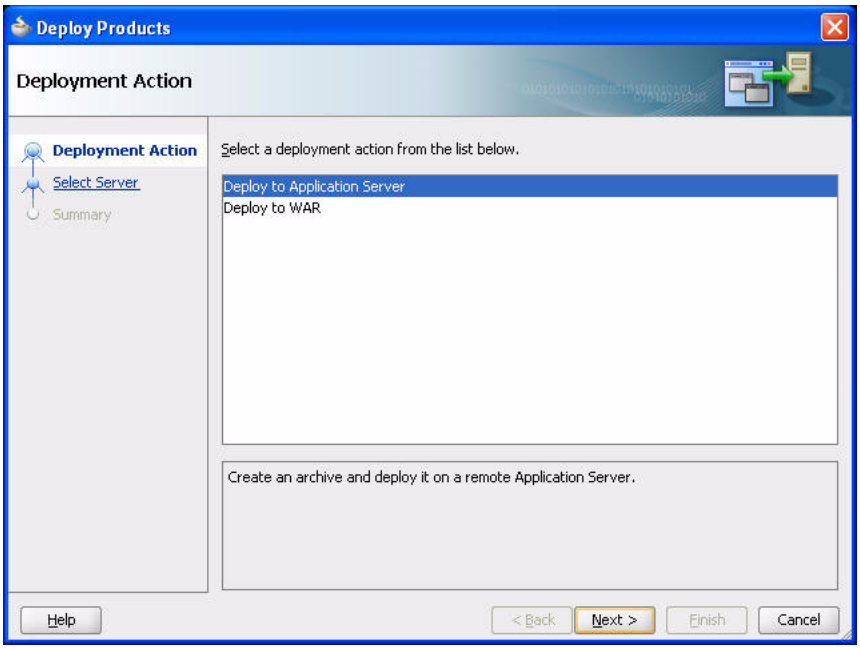
10. In the Create Deployment Profile -- EAR File dialog, in the Deployment Profile Name field, enter `Products` (Figure 6–40), then click **OK**.

Figure 6–40 Creating a Deployment Profile



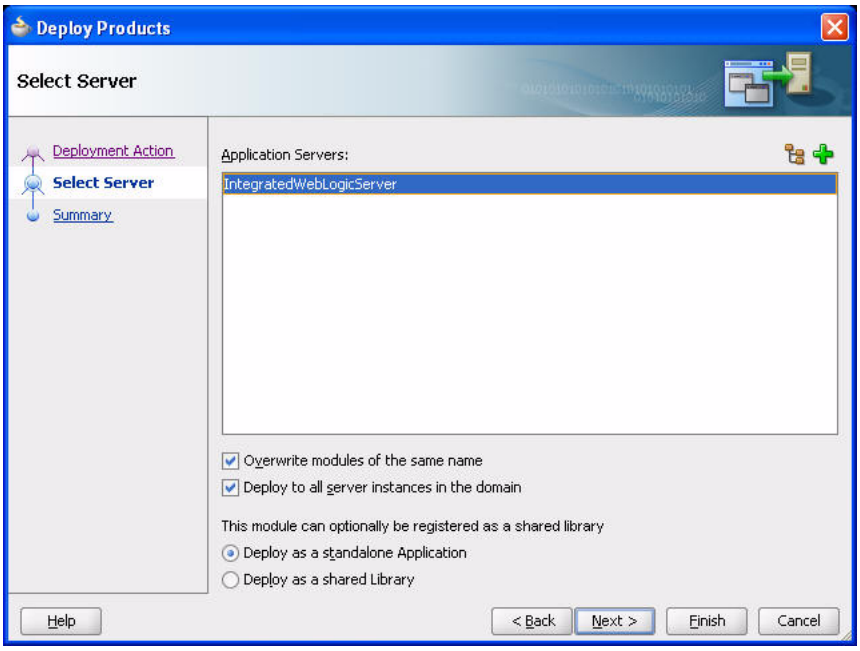
11. Click **OK**, then click **OK** again.
12. Now, we're ready to deploy our standards-based portlet. In the Application Navigator, right-click the **Portlets** project, and choose **Deploy,** then **Products** from the context menu.
13. In the Deploy Products wizard, on the Deployment Action page, select **Deploy to Application Server**, then click **Next** (Figure 6–41).

Figure 6–41 Deployment Action page of the Deploy Products Wizard



14. On the Select Server page of the Deploy Products wizard, select **IntegratedWebLogicServer**, then click **Next** (Figure 6–42).

Figure 6–42 Select Server Page of the Deploy Products Wizard



15. On the Summary page, ensure that the Application Server listed is the IntegratedWebLogicServer and that the Archive Details shows the Output file pointing to *JDEV_USER_HOME*`\jdeveloper\mywork\MyTutorialPortlet\Portlets\deploy\Products.war`.

16. Click **Finish**.

17. If the Select deployment type dialog displays, leave the default options and click **OK** (Figure 6–43).

Figure 6–43 Select Deployment Type

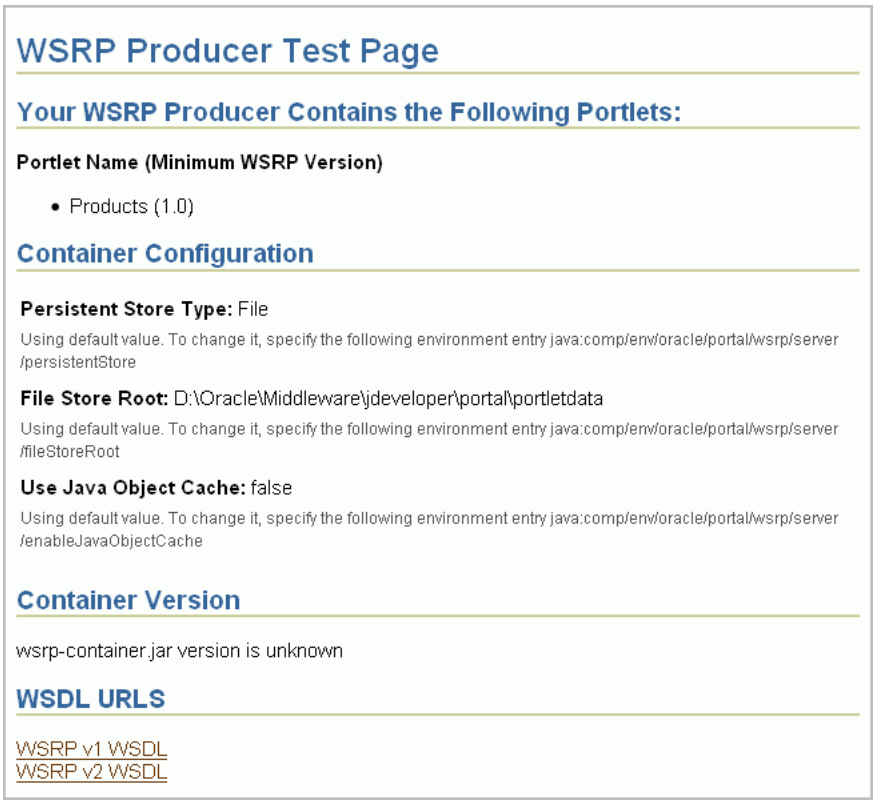
Verify that the deployment is complete by checking the Deployment - Log below the Visual Editor (the Design view) for the message "Deployment finished."

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

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

The WSRP Producer Test Page displays as shown in Figure 6–44.

Figure 6–44 WSRP Producer Test Page



19. 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 6–45).

Figure 6–45 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`.

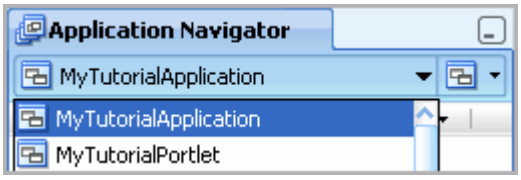
Step 5: Register the Standards-Based Portlet with Your Application

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:

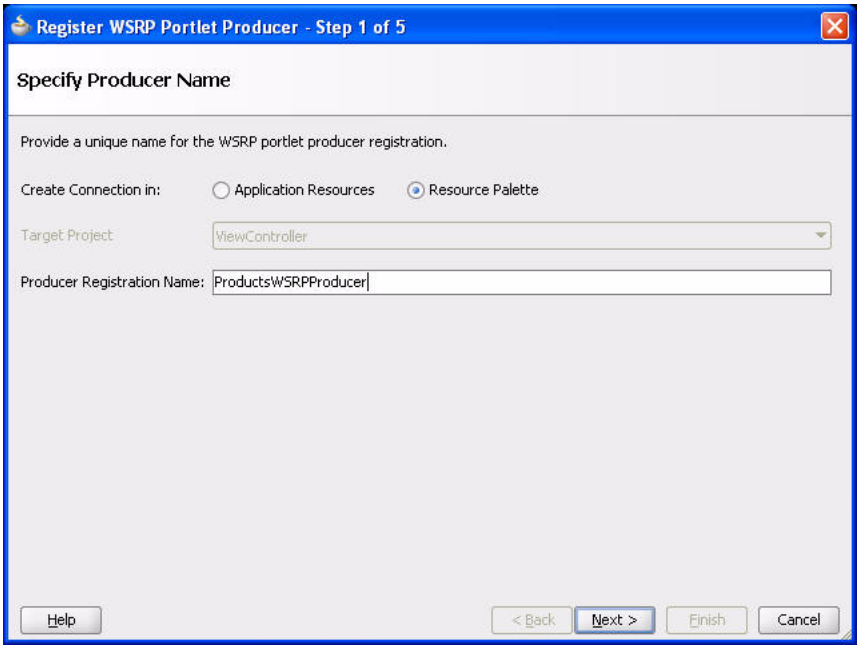
1. In the Application Navigator, choose **MyTutorialApplication** from the list to return to our custom WebCenter application (Figure 6–46).

Figure 6–46 MyTutorialApplication in the Application Navigator List



2. In the Resource Palette, click the folder icon, then choose **New Connection** and **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 6–47).

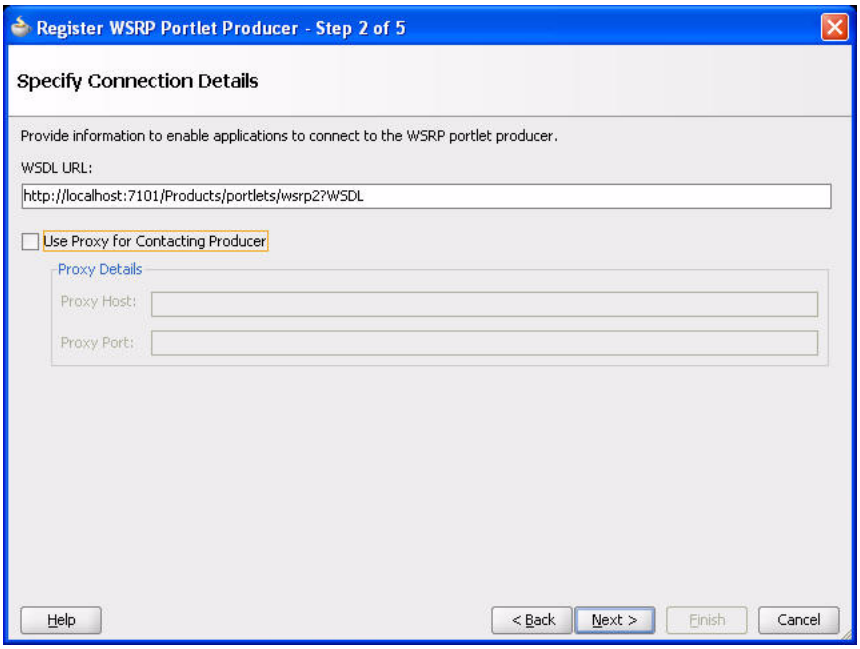
Figure 6–47 Register WSRP Portlet Producer -- Name



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

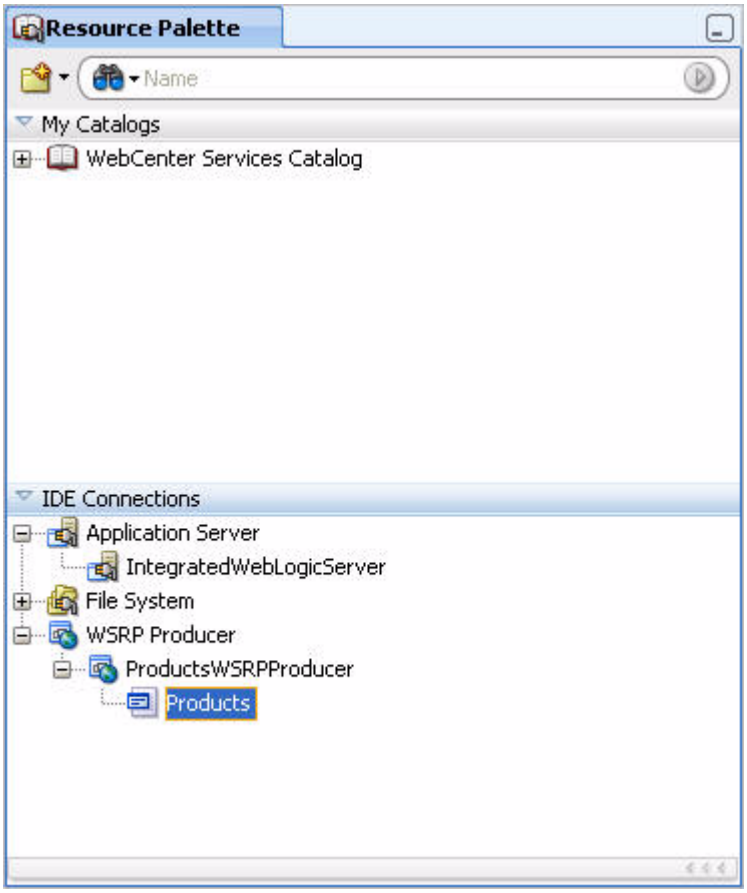
```
http://localhost:7101/Products/portlets/wsrp2?WSDL
```

Figure 6–48 Register WSRP Portlet Producer -- Connection



5. Since our Integrated WebLogic 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 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 6–49 shows the **Products** portlet here.

Figure 6–49 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.

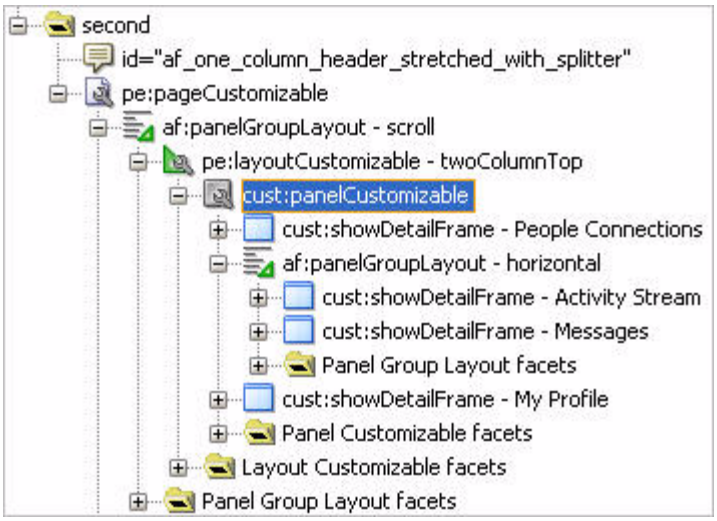
Step 6: Test the Standards-Based Portlet in Your Application

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, Panel Group Layout, Layout Customizable,** and **Panel Customizable**, as shown in Figure 6–50. You should see the Show Detail Frame called My Profile. You will add the portlets below this component.

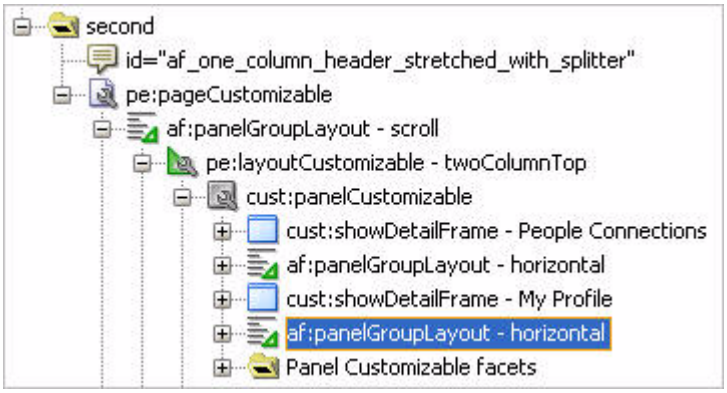
Note: As previously mentioned, you can use the pushpin in the Structure window to freeze the current view. For this step, you should click **MyPage** in the Design view, then ensure the pushpin is in the "freeze" position (pressed).

Figure 6–50 Panel Customizable Where You Will Add the Products Portlet



3. In the Component Palette, choose **ADF Faces** from the list.
4. Drag and drop a **Panel Group Layout** just below the Show Detail Frame called My Profile.
5. Set the Layout property to **horizontal** and the StyleClass (located below Style and Theme) to `AFStretchWidth`. Figure 6–51 shows the new Panel Group Layout in the Structure Window.

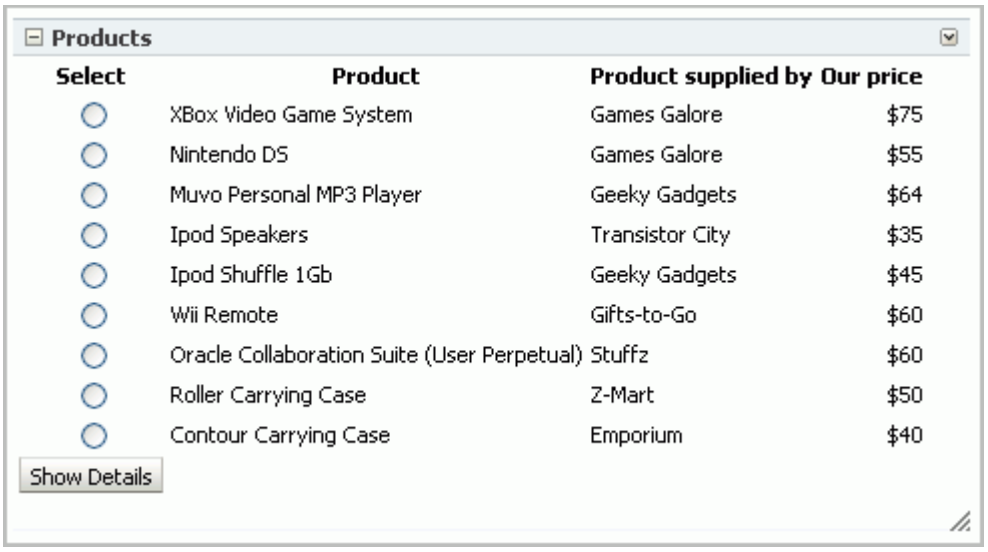
Figure 6–51 New Panel Group Layout



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

   You will 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 6–52 shows the Products portlet at runtime.

Figure 6–52 Products Portlet in a Browser Window



For more information about creating standards-based JSR 168 portlets and using them with a WebCenter application, see Part VIII, "Working with Portlets and Portals" in *Oracle Fusion Middleware Developer's Guide for Oracle WebCenter*.

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 WebLogic Server, also referred to as the Integrated WebLogic Server. Be sure you have followed the steps in Chapter 2, "Preparing for the Tutorial" before you proceed.

To register the preconfigured portlet producer:

1. Ensure the Integrated WebLogic Server is running.
2. To find out the URL for the OmniPortlet producer running on the Integrated WebLogic Server, choose **WebCenter Preconfigured Server Readme** from the Help menu. This Read Me file shows all the information needed to use the portlets that are predeployed to the Integrated WebLogic 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:

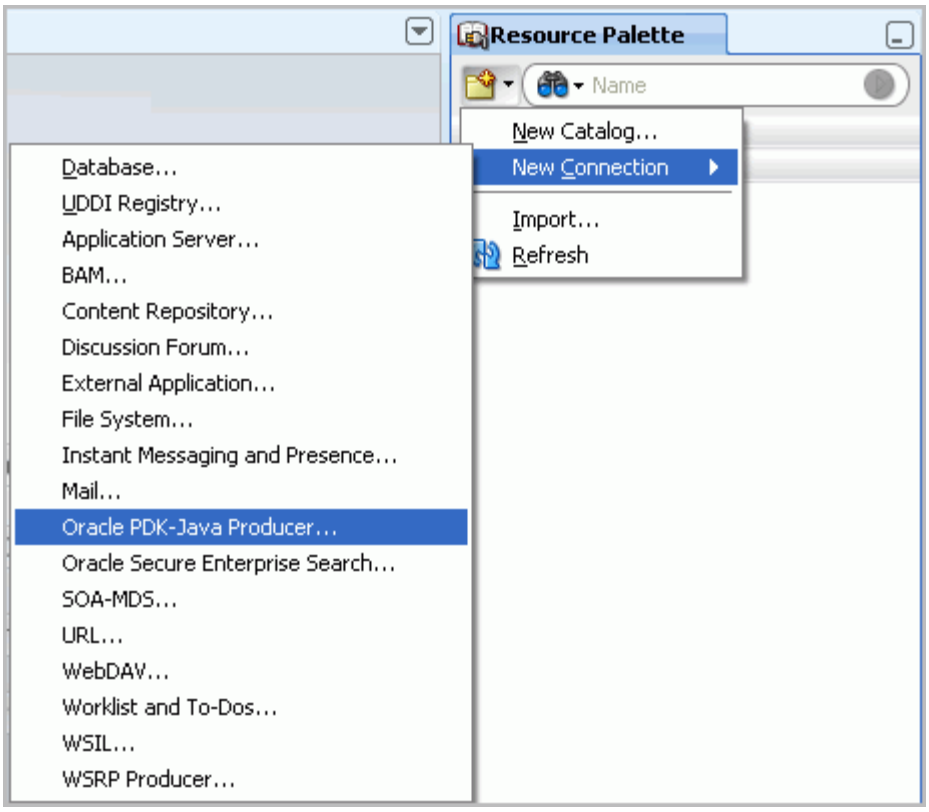
   ```
   http://127.0.0.1:7101/portalTools/omniPortlet/providers/omniPortlet
   ```

   Entering this URL in your browser displays the OmniPortlet producer test page (Figure 6–53).

Figure 6–53 OmniPortlet Producer Test Page

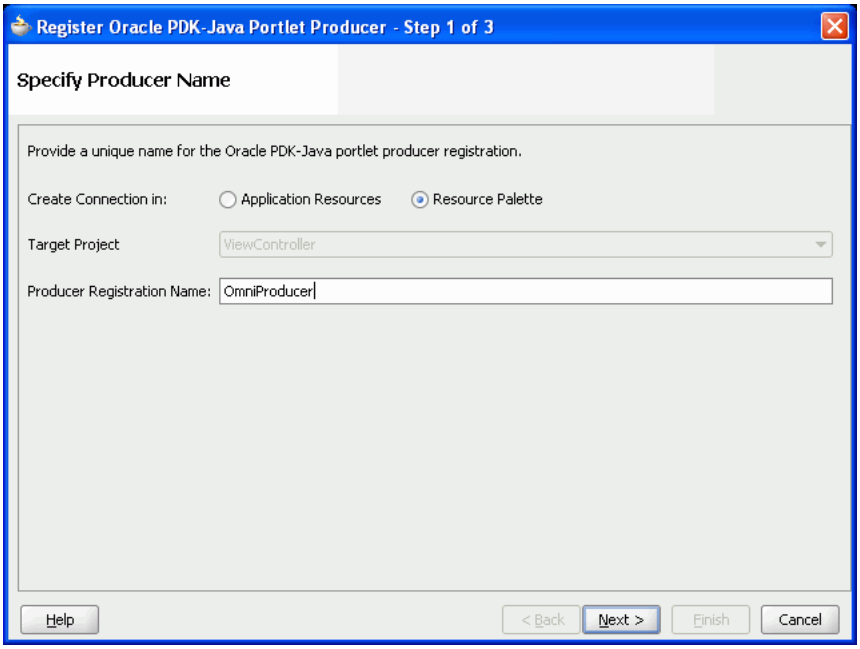
5. In the Resource Palette, click the **New** icon next to the search toolbar, then choose **New Connection**, and then **Oracle PDK-Java Producer** (Figure 6–54).

Figure 6–54 Registering an Oracle PDK-Java Producer



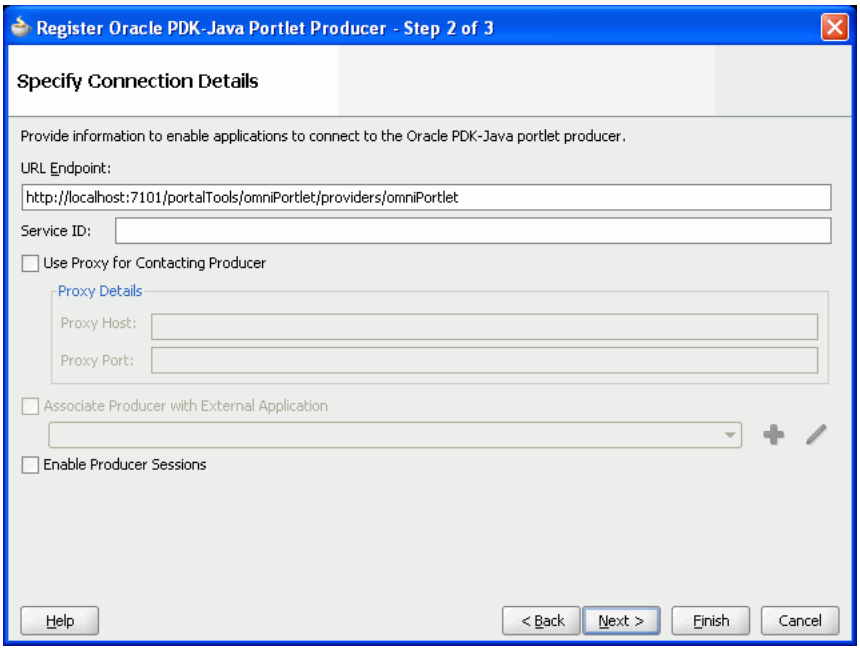
6. On Step 1 of the Register Oracle PDK-Java Portlet Producer wizard, let's enable this portlet producer to be used across all our applications. Select **Resource Palette**.
7. In the Name field, enter `OmniProducer` (Figure 6–55).

Figure 6–55 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 6–56).

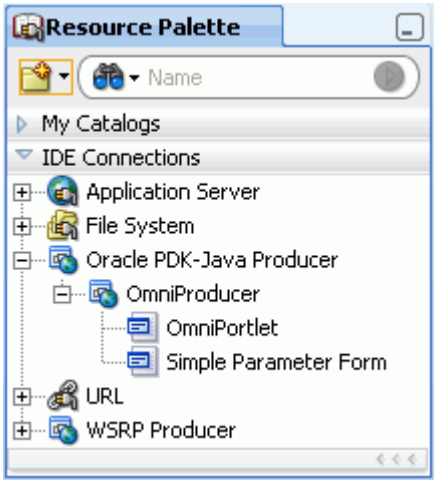
Figure 6–56 Specifying the Connection Details for the OmniPortlet Producer



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 6–57.

Figure 6–57 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 Integrated WebLogic Server. It lets you quickly create portlets using a variety of default layouts and data sources.

Figure 6–58 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.

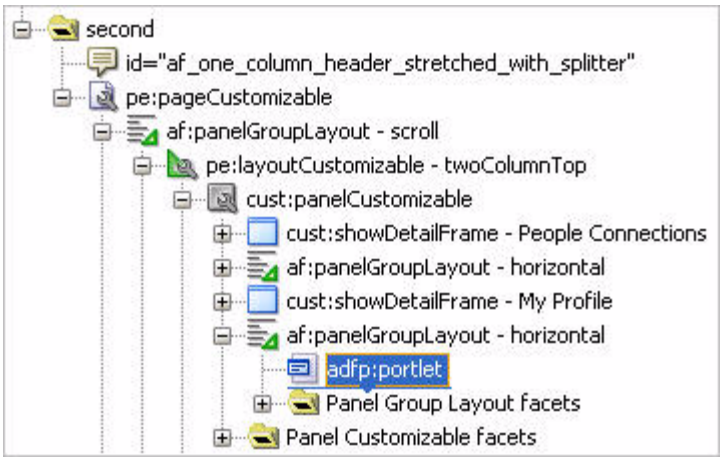
Figure 6–58 OmniPortlet at Runtime

Before you can use this OmniPortlet, ensure that you have added the sample schema to your database and that you have started the Integrated WebLogic 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 horizontal Panel Group Layout where you added the Products portlet.

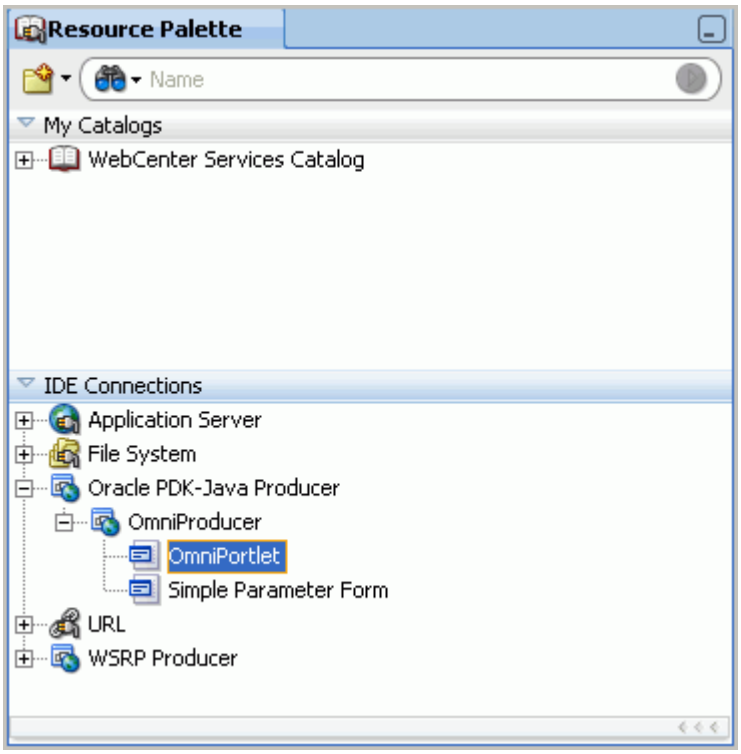
Note: As previously mentioned, you can use the pushpin in the Structure window to freeze the current view. For this step, you should click **MyPage** in the Design view, then ensure the pushpin is in the "freeze" position (pressed).

Figure 6–59 Panel Group Layout Containing the Products Portlet



4. In the Resource Palette, under IDE Connections, navigate to **Oracle PDK-Java Producer, OmniProducer,** and **OmniPortlet** (Figure 6–60).

Figure 6–60 OmniPortlet in the Resource Palette



5. Drag and drop **OmniPortlet** onto your page onto the **Panel Group Layout**. It should display below the existing **adfp:portlet** in the Structure window. If you see an error message, ensure you have started the Integrated WebLogic Server, then try again.

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

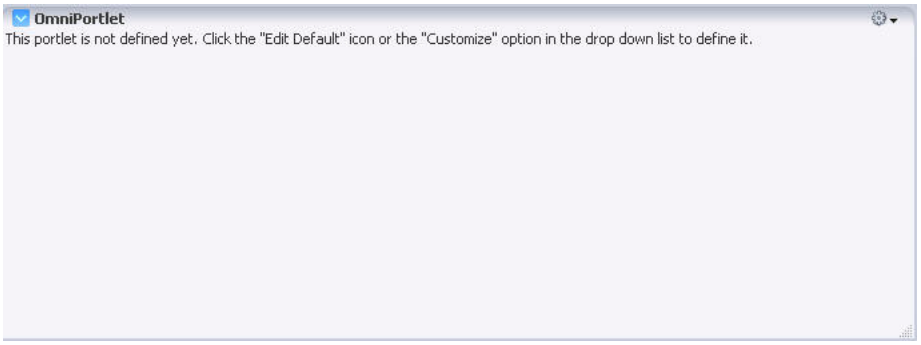
6. Set the **Height** property for the portlet to `230px`.

7. Now that we've added OmniPortlet to our page, let's customize its contents.

Run your page to the browser and log in as `Lisa/welcome1`.

Figure 6–61 shows the OmniPortlet on MyPage in your browser.

Figure 6–61 Undefined OmniPortlet at Runtime



Step 9: Define OmniPortlet at Runtime

In this section, we will 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 6–63).

Figure 6–62 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 6–7. 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.

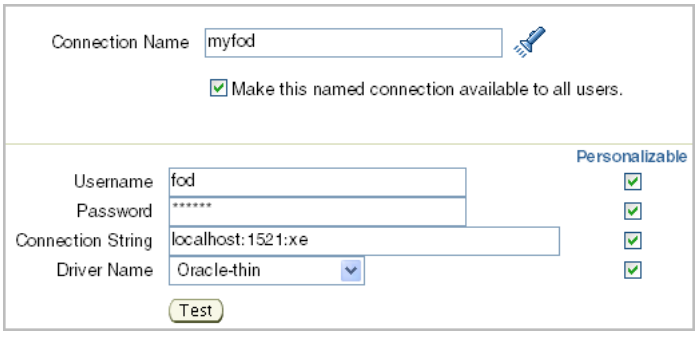
Example 6–7 OmniPortlet SQL Statement

```
SELECT distinct  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; localhost also points to the IP address 127.0.0.1. Notice that you can also click the **Test** button to see what the statement will return.

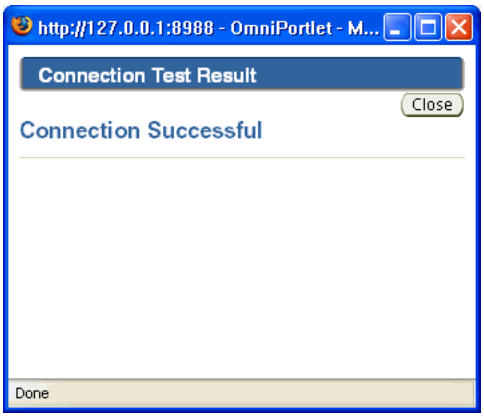
4. Under Connection, click **Edit Connection**.
5. Enter a name for your connection and the connection information for your database. The schema username is `fod` and the password is `fusion`. For example, if you are using an Oracle XE database locally, the page would look like Figure 6–63.

Figure 6–63 Connection Information for OmniPortlet



6. Let's make sure the connection information is correct by clicking **Test**. Figure 6–64 shows a successful test message.

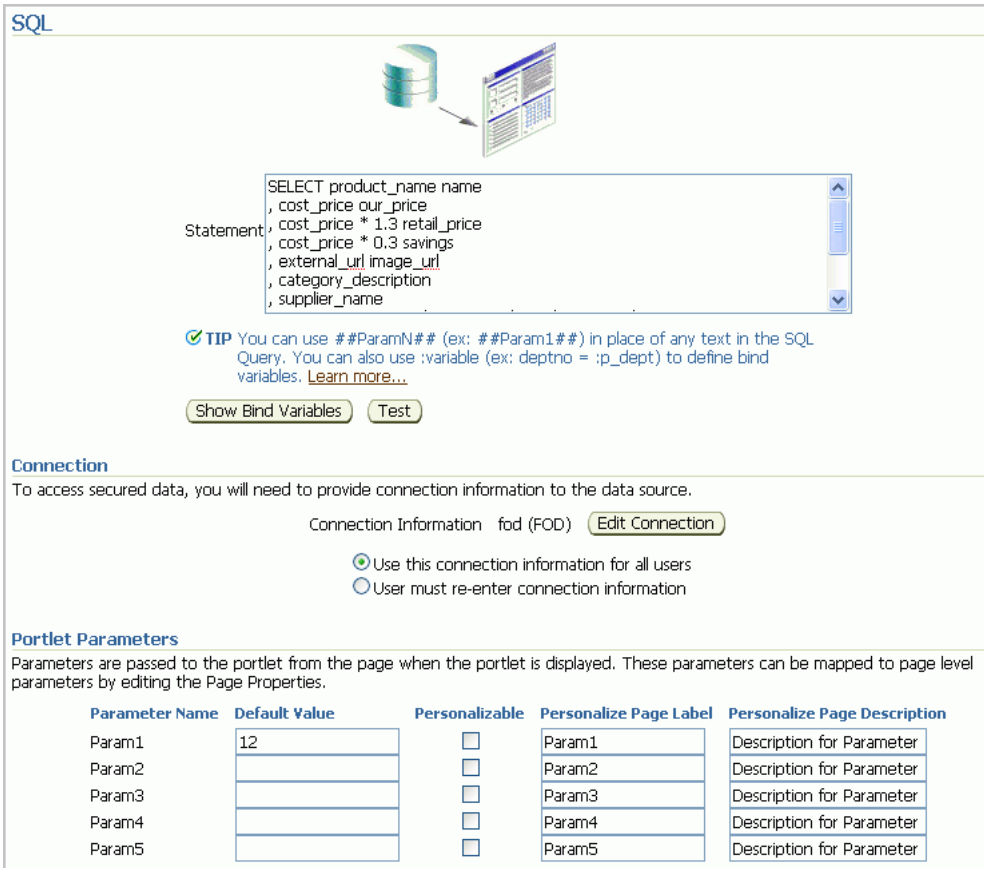
Figure 6–64 Successful Connection Message



7. Click **Close**, then click **OK** to finish creating the connection.
8. 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 6–65 shows the SQL Source page.

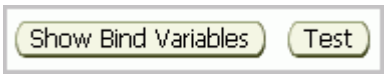
Figure 6–65 SQL Source Page



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 Resource Files to the Application, but you must run the application to your browser again in order for OmniPortlet to recognize the new folder.

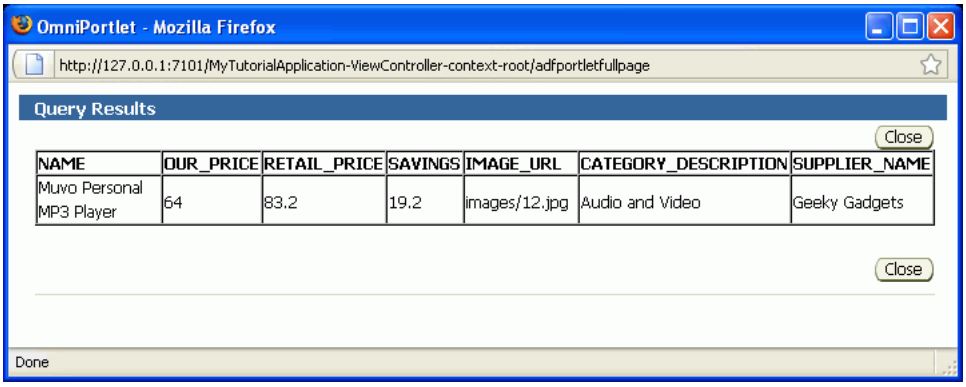
9. Click **Test** next to the Show Bind Variables button to validate the SQL statement and connection.

Figure 6–66 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.

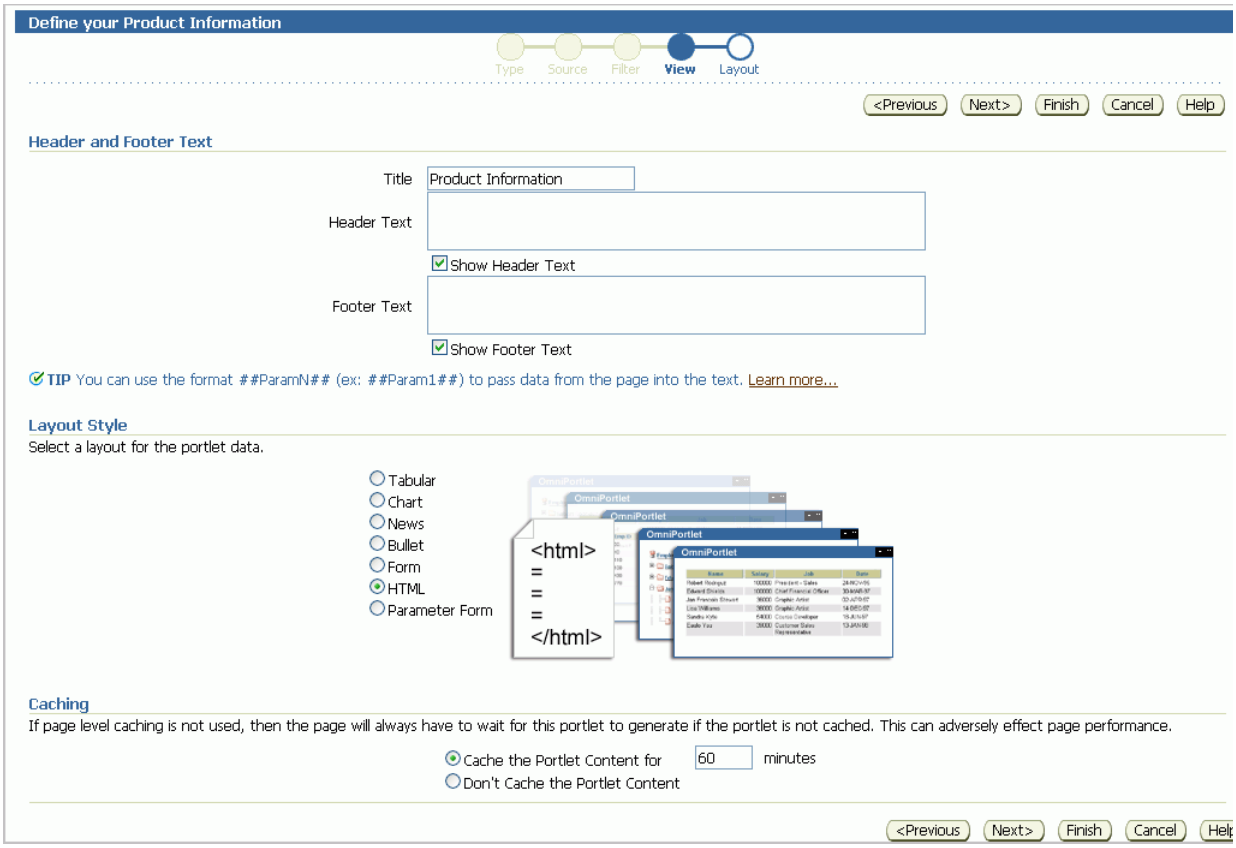
Figure 6–67 SQL Statement Test Results



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 6–68, then click **Next**.

Figure 6–68 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 6–8 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.

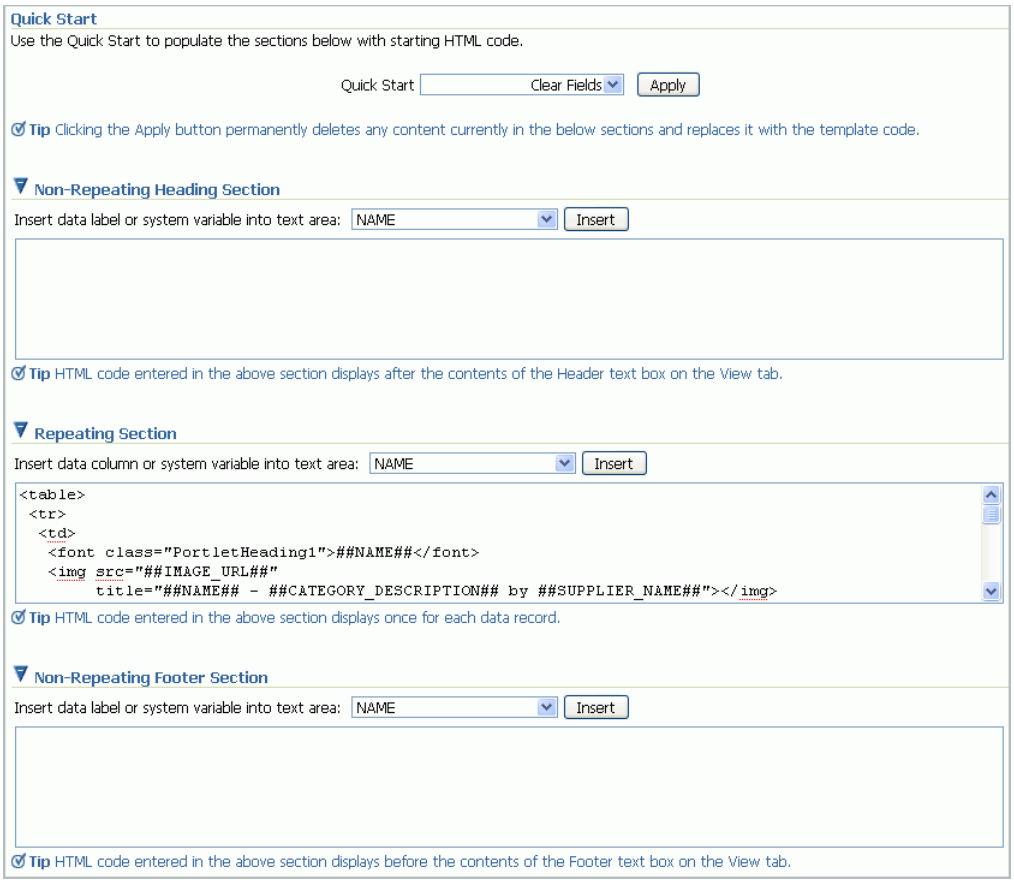
Example 6–8 OmniPortlet HTML Layout Code

```
<table>
 <tr>
  <td>
   <font class="PortletHeading1">##NAME##</font>
   <img src="##IMAGE_URL##"
        title="##NAME## - ##CATEGORY_DESCRIPTION## by ##SUPPLIER_NAME##"></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> </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 6–69.

Figure 6–69 HTML Layout Page



16. Click **Finish**. Figure 6–70 shows the OmniPortlet in your browser.

Figure 6–70 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

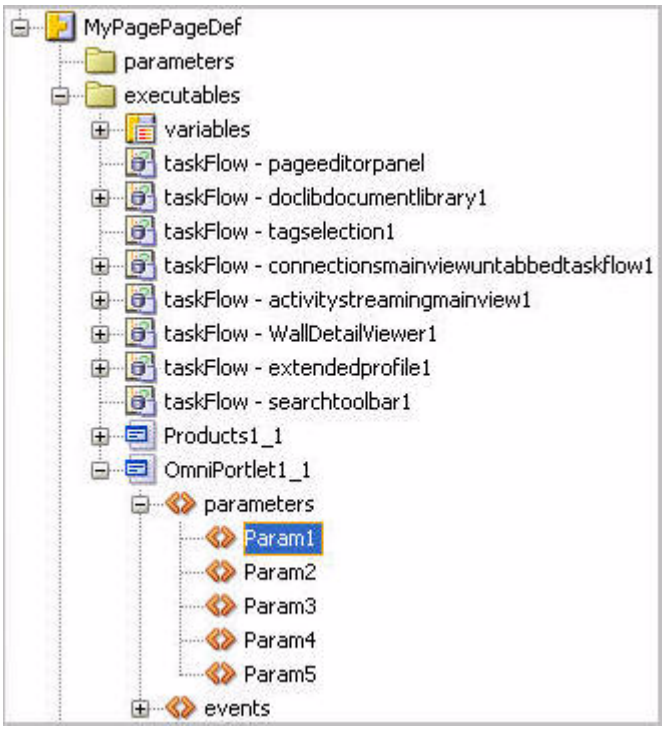
When you added the code for the standards-based portlet called `Products`, you also included a parameter called `productId`. When you select different options in the `Products` portlet, the application will send this parameter to OmniPortlet so that it can display the details of a particular product 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**. You created the page definition for MyPage in "Step 3: Add ADF Security Policies to Your Application" in Chapter 4, "Adding Security to Your Application."
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** then parameters, then select the portlet variable **Param1**, as shown in Figure 6–71.

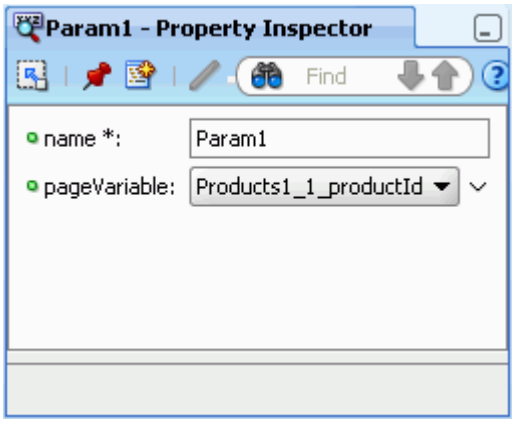
Figure 6–71 OmniPortlet Variable in the Structure Window



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

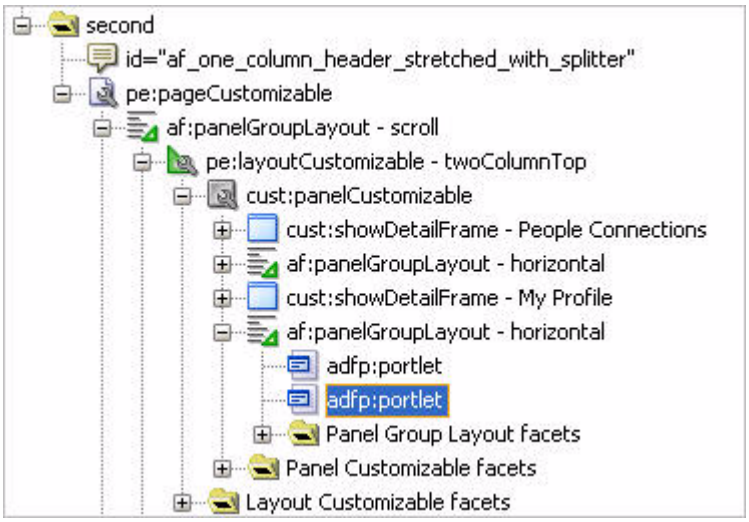
   Set the **pageVariable** property to **Products1_1_productId**, as shown in Figure 6–72.

Figure 6–72 Setting the Default Value



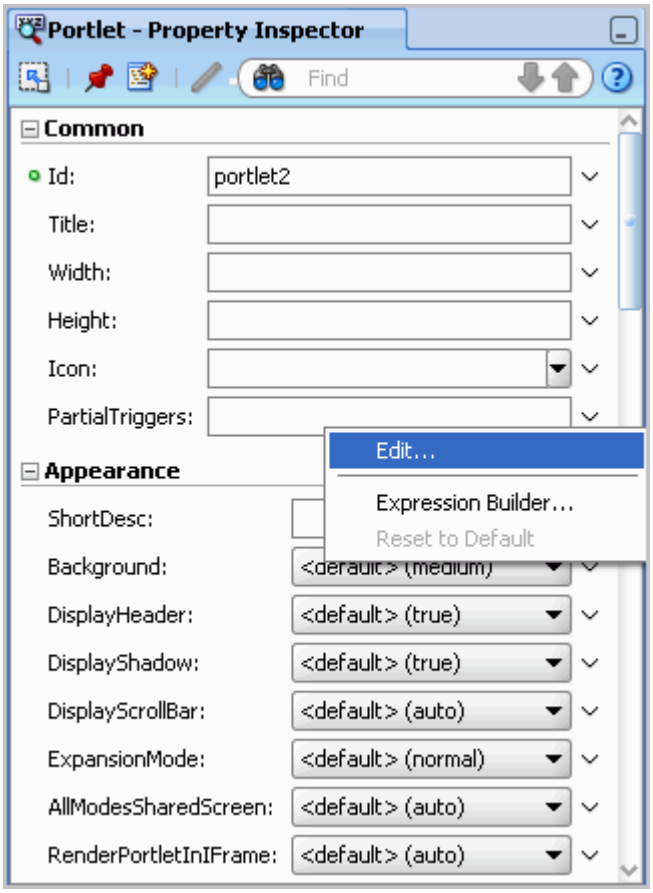
6. Now we want the detail portlet (OmniPortlet) to refresh whenever the value from the master portlet changes. To do so, we add a Partial Trigger to the detail portlet. Click the **MyPage.jspx** tab at the top of the Visual Editor to bring it into focus.
7. Select the **OmniPortlet** on the page in the Structure window. The portlet is the second instance of `adfp:portlet` in the Panel Group Layout, as shown in Figure 6–73.

Figure 6–73 OmniPortlet on MyPage in the Design View



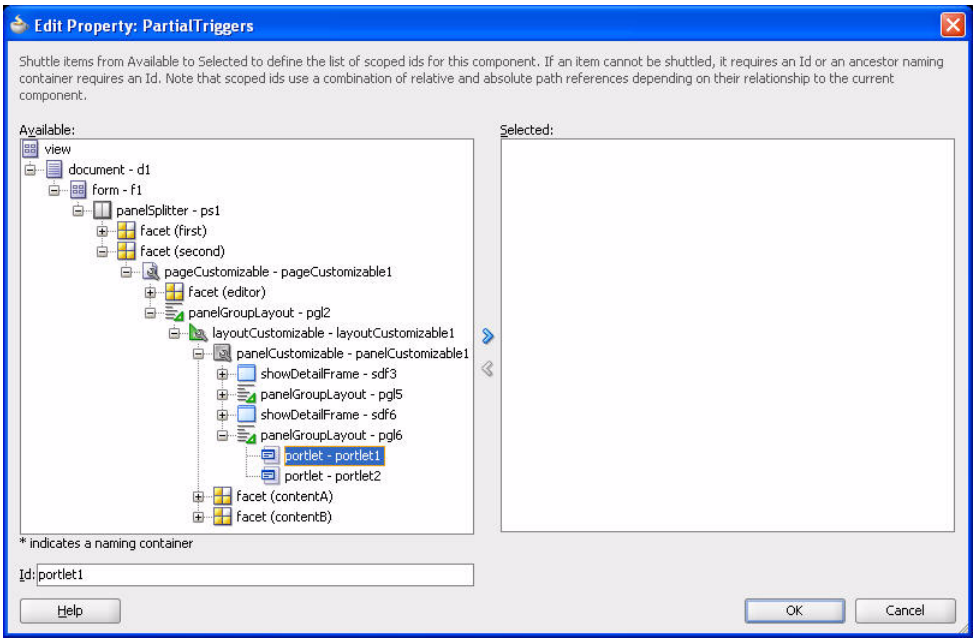
8. In the Property Inspector for OmniPortlet, under **Common**, click the arrow next to **PartialTriggers**, then choose **Edit** (Figure 6–74).

Figure 6–74 Editing the PartialTriggers Property for OmniPortlet



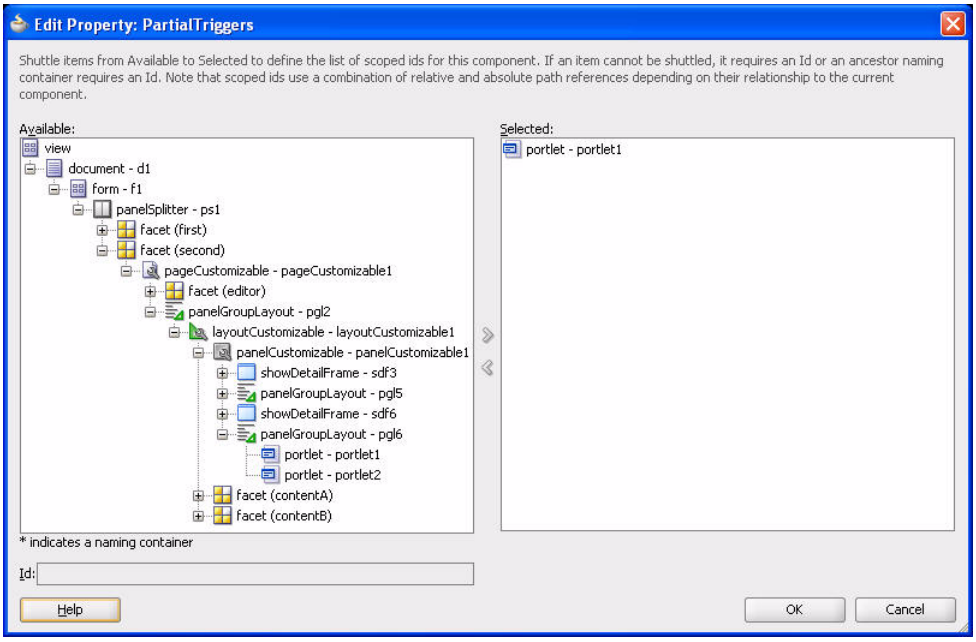
9. In the Edit Property: PartialTriggers dialog, the Portlet ID for OmniPortlet (`portlet - portlet2`) is automatically selected. Locate the **Portlet ID** for the Products portlet, which is directly above the OmniPortlet (Figure 6–75).

Figure 6–75 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 6–76).

Figure 6–76 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 and log in as `Lisa/welcome1`. In the next step, we will test how the JSR 168 (Products) portlet and the OmniPortlet interact at runtime.

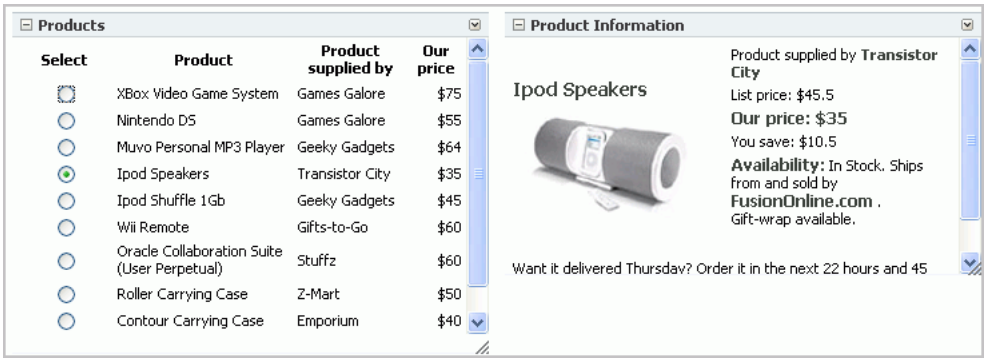
Step 11: Test the Interaction Between the Portlets

Let's test the portlets at runtime.

To test the interaction between the portlets:

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 6–77).

Figure 6–77 Testing the Interaction Between the Portlets



For more information about creating portlets and using them with a WebCenter application, see Part VIII, "Working with Portlets and Portals" in *Oracle Fusion Middleware Developer's Guide for Oracle WebCenter*.

Now that you have created two portlets and added them to your application, let's see how we can change the look and feel of our application using a skin, as well as personalize the application at runtime in Chapter 7, "Changing the Look and Feel of Your Application."

7

Changing the Look and Feel of Your Application

Oracle WebCenter enables you design the look and feel of your application specifically for your organization at both design time (which you, as a developer, can do before deploying the application) and at runtime (which the application administrator or user can do after you have deployed the application).

At design time in JDeveloper, Oracle WebCenter enables you to use a "skin," which is essentially a global style sheet (based on CSS) that you can apply to your entire application. Once you apply the skin to your application, every layout component automatically uses the styles assigned by the skin. You cannot change this skin at runtime or post-deployment.

At runtime in your browser, Oracle WebCenter enables authenticated users (administrators, users, and so on) to *personalize* their view of their application using Oracle Composer. For example, if our sample user Lisa logs into the application at runtime, she can add, remove, or change the appearance of components or services that are available to the application. Only Lisa can view these modifications; if Dan or Alex log in, they see their own personal view of the application.

In this lesson, you, as a developer, will change the look and feel of your application at design time that all users can view at runtime. Then, to understand how your users can view and personalize the application, you will then log in as different users based on the roles you set in Chapter 4, "Adding Security to Your Application," and personalize the view of the application for that user at runtime.

Figure 7–1 shows how the application looks with the new skin.

Figure 7–1 Partial View of MyPage at Runtime with the New Skin

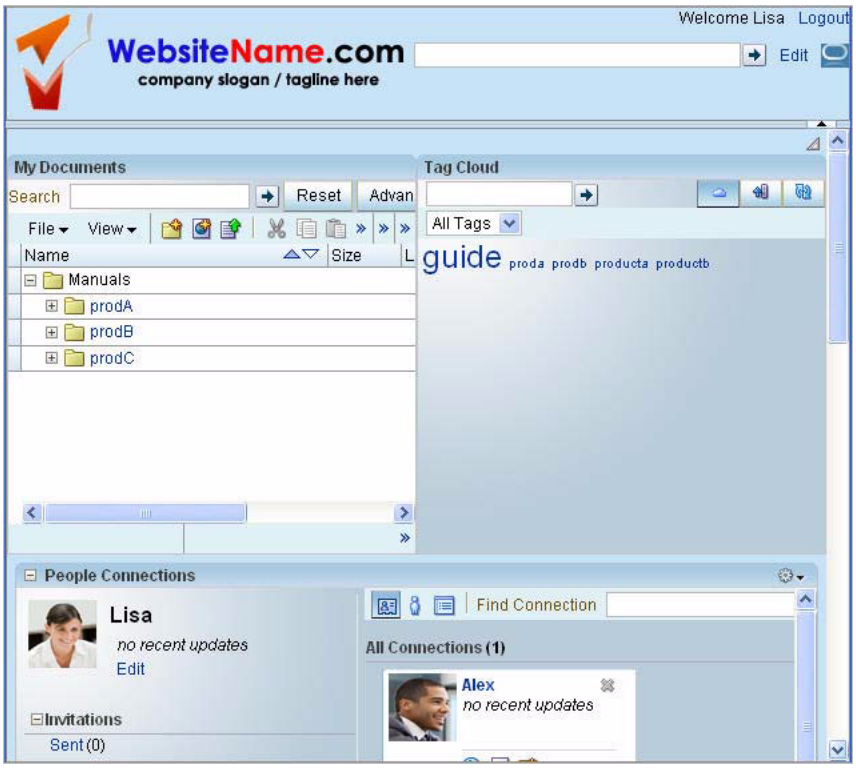


Figure 7–2 shows a partial view of Lisa's personalized page.

Figure 7–2 Partial View of Lisa's Personalized Page

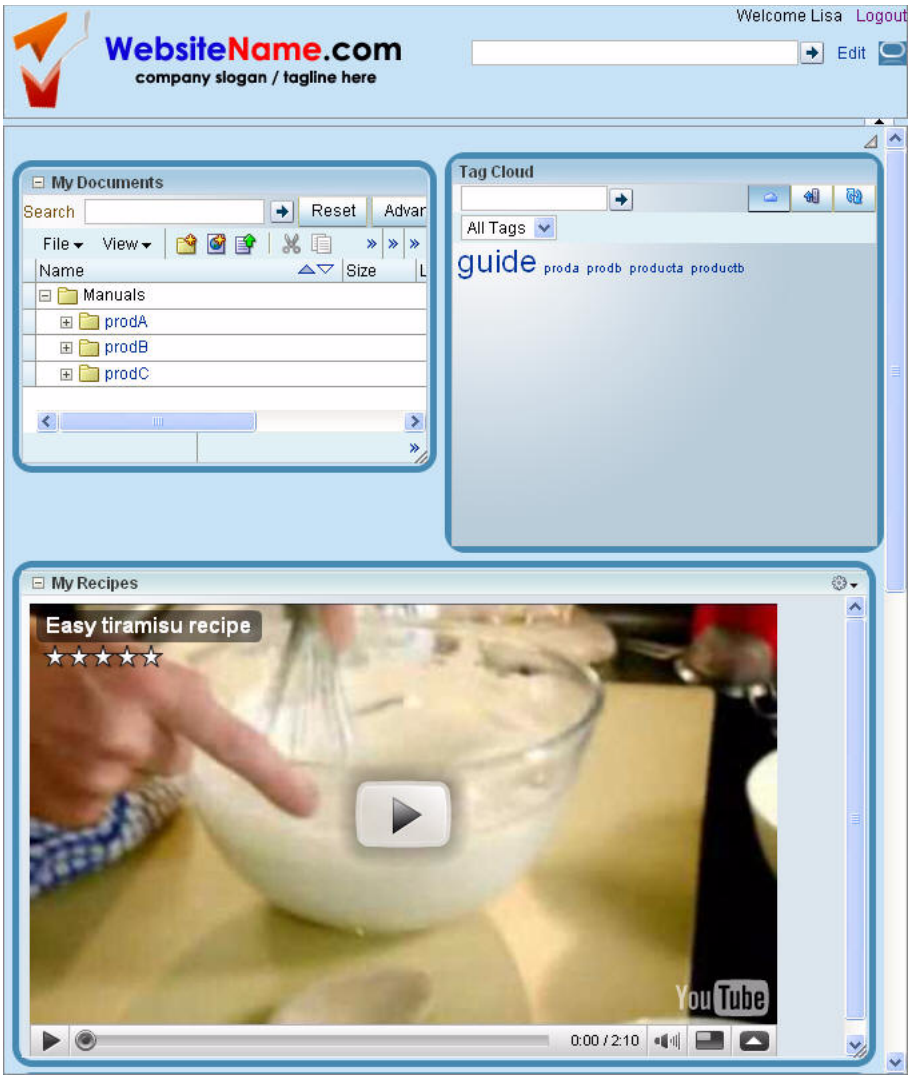


Figure 7–3 shows a partial view of Alex's personalized page.

Figure 7–3 Partial View of Alex's Personalized Page

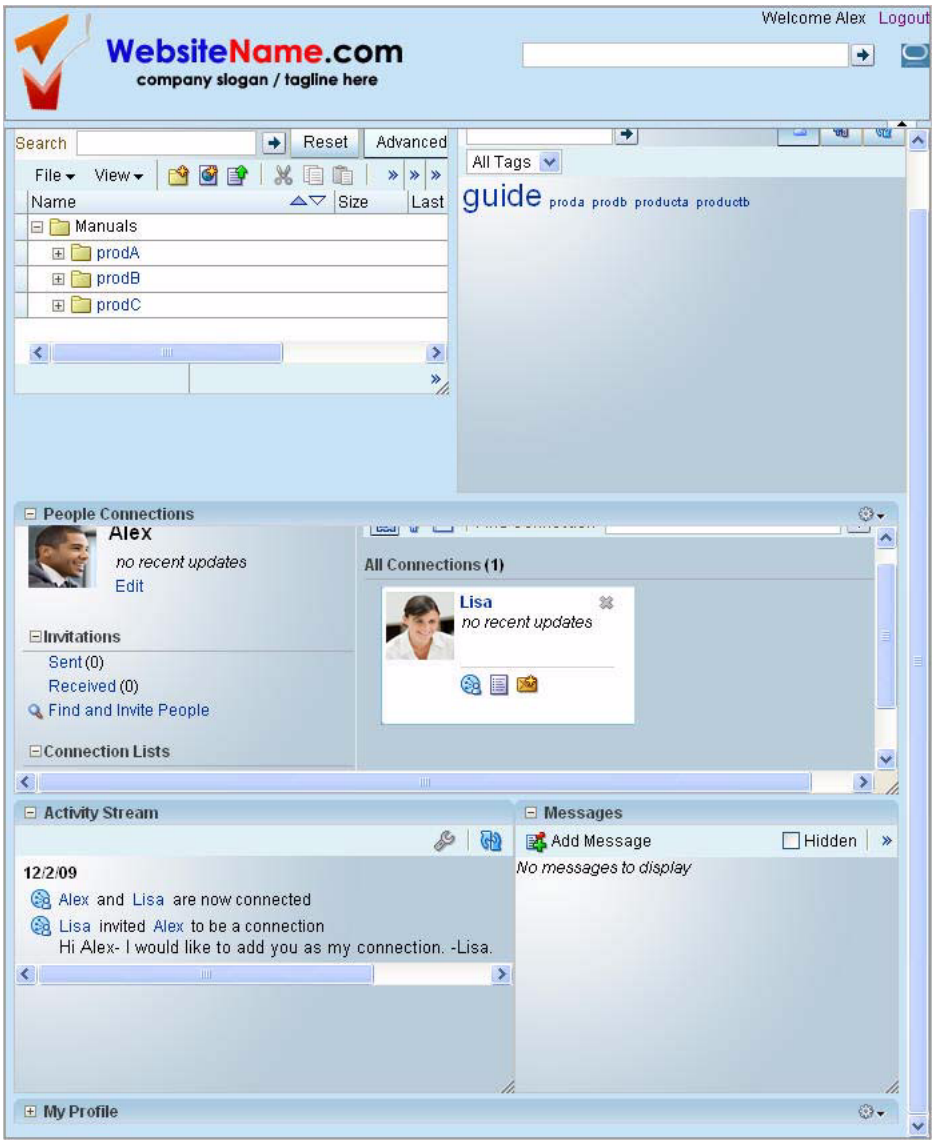
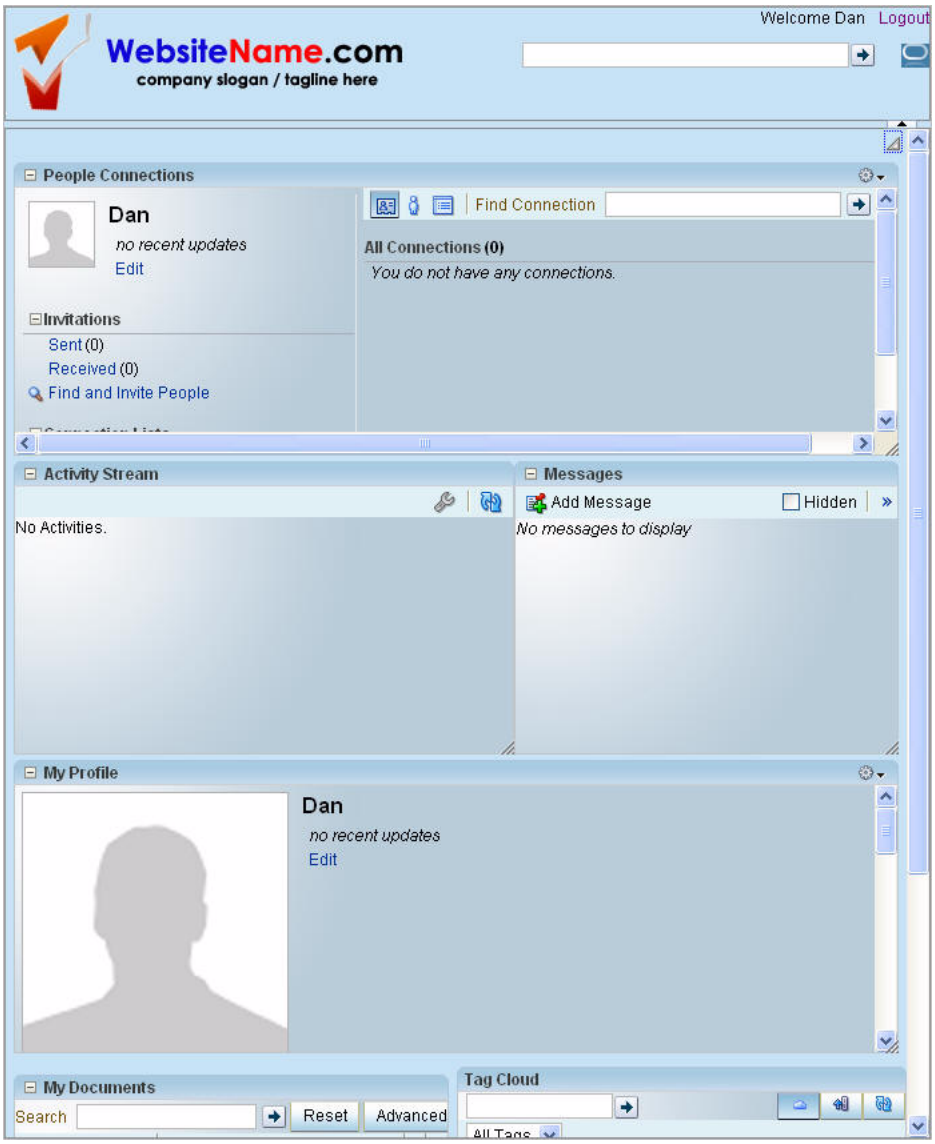


Figure 7–4 shows Dan's personalized page.

Figure 7–4 New Layout on Dan's Personalized Page



Introduction

This lesson contains the following steps:

- Step 1: Change the Application Look and Feel Using a Skin
- Step 2: Personalize One User's (Lisa's) Page
- Step 3: Personalizing a Second User's (Alex's) Page
- Step 4: Personalizing a Third User's (Dan's) Page

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

Step 1: Change the Application Look and Feel Using a Skin

When you create a custom WebCenter application using the WebCenter Application template, the application includes a skin by default. You can extend this default skin with your own CSS file to change the look and feel of your application.

In this step, you will extend the default skin to change the look and feel of your application. First, you can either create a new CSS file or use an existing one; we've provided a CSS file you can use in this step. Next, you will create an XML file that tells the application to extend its default skin and use the CSS file you created to modify the appearance of the pages in your application. Finally, you will copy the CSS file into the directory that the XML file references.

Ensure you've followed the steps for adding the `css` and `images` folders from the Tutorial Sample Files to your application in Chapter 3, "Creating a WebCenter Application with a Customizable Page."

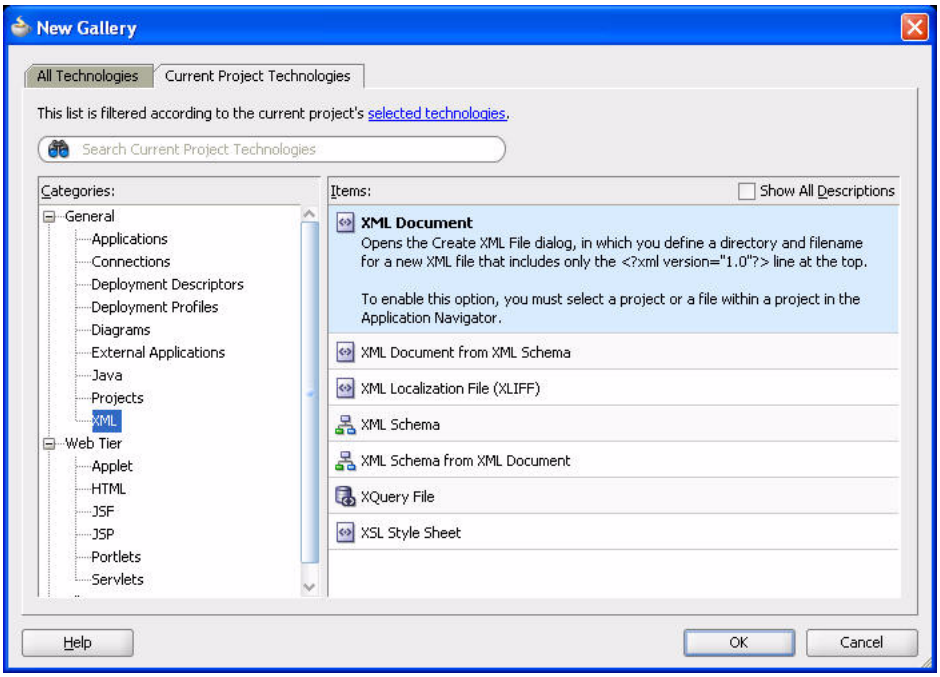
To change the application skin:

1. First, we need to create a configuration file, called `trinidad-skins.xml`, to tell the application to extend the default skin.

   Make sure your page, `MyPage.jspx`, is open in JDeveloper and that `MyTutorialApplication` is in focus in the Application Navigator.

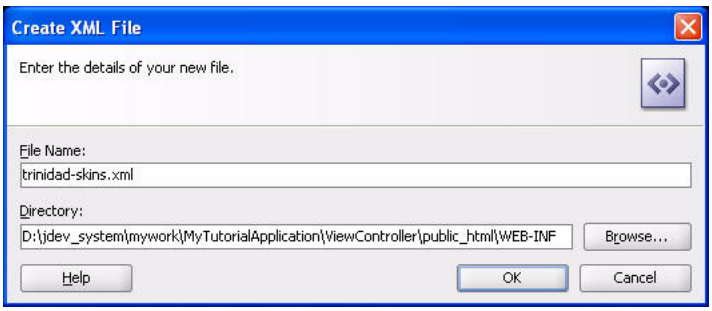
2. In the Application Navigator, under the ViewController project, open **Web Content.**
3. Right-click **WEB-INF**, then choose **New** to display the New Gallery.
4. In the New Gallery, under General in the left pane, select **XML**
5. In the right pane, select **XML Document**, then click **OK** (Figure 7–5).

Figure 7–5 Creating a New XML Document



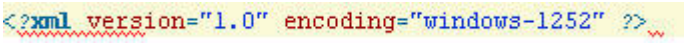
6. In the Create XML File dialog, in the File Name field, enter `trinidad-skins.xml`, then click **OK** (Figure 7–6). The new XML file displays in the Design view.

Figure 7–6 Creating the trinidad-skins.xml File



7. Delete the code in this editor, as shown in Figure 7–7.

Figure 7–7 Deleting the Default Code in the trinidad-skins.xml File



8. Paste the following code snippet into the `trinidad-skins.xml` file (Example 7–1):

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

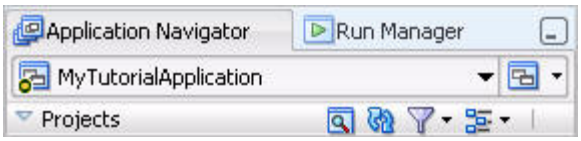
Example 7–1 Skin Code in the trinidad-skins.xml File

```
<?xml version="1.0" encoding="windows-1252" ?>
<skins xmlns="http://myfaces.apache.org/trinidad/skin">
       <skin>
              <id> TutorialSkin.desktop</id>
              <family>TutorialSkin</family>
              <extends>blafplus-rich.desktop</extends>
              <render-kit-id>org.apache.myfaces.trinidad.desktop</render-kit-id>
              <style-sheet-name>css/TutorialSkin.css</style-sheet-name>
       </skin>
</skins>
```

9. By creating `trinidad-skin.xml` file containing the code in Example 7–1, you created a new skin called `TutorialSkin`, which extends a default skin contained in the application called `blafplus-rich`. Every skin defined in this file must have a corresponding CSS file. In this case, the CSS file is called `TutorialSkin.css`. You already added the `TutorialSkin.css` file when you added the images and other resources to your application in Chapter 3, "Creating a WebCenter Application with a Customizable Page."

10. Refresh the Application Navigator by clicking the **Refresh** icon next to the Projects list (Figure 7–8).

Figure 7–8 The Refresh Icon in the Application Navigator



11. In the Application Navigator, under **ViewController**, **WEB-INF**, open the **trinidad-config.xml** file.
12. Find the following code:

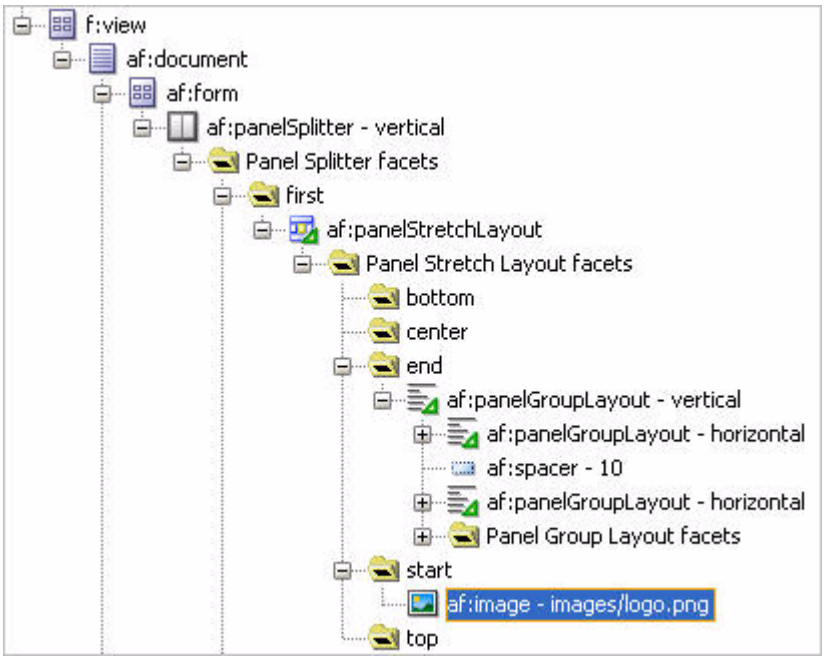
```
<skin-family>fusion</skin-family>
```

13. In the code, replace `fusion` with `TutorialSkin` so the code looks like this:

```
<skin-family>TutorialSkin</skin-family>
```

14. Save the file.
15. Restart JDeveloper to apply your changes to the `trinidad-skins.xml` and `trinidad-config.xml` files.
16. Bring **MyPage.jspx** into focus.
17. In the Structure window for `MyPage.jspx`, navigate to the **start** facet that contains the `logo.png` file you added in Chapter 3, "Creating a WebCenter Application with a Customizable Page." Figure 7–9 shows the logo file in the Structure window.

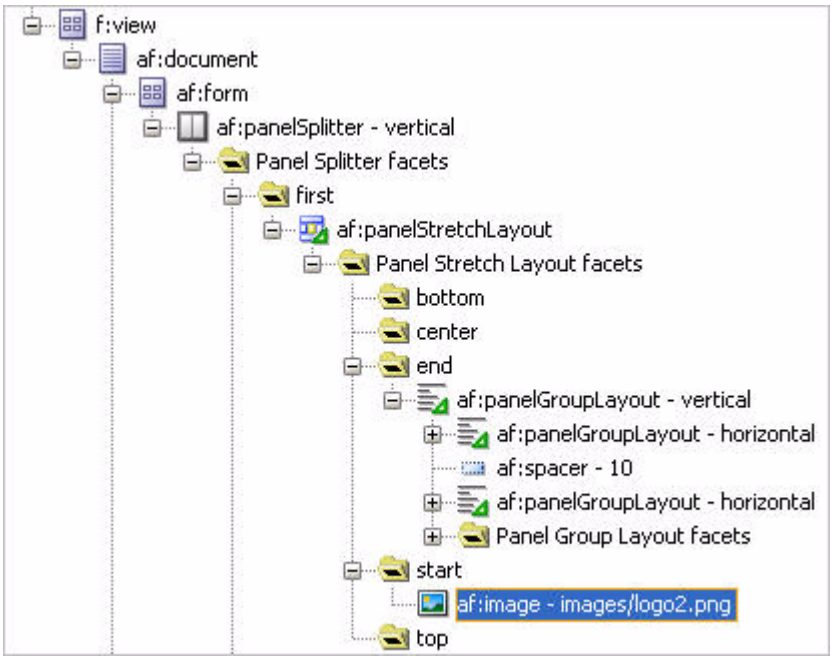
Figure 7–9 Logo.png in the start facet



18. Replace this file with the `logo2.png` file located in the **images** directory. To do so, delete the `af:image - images/logo.png` from the Structure window (while it is selected, press the Delete key).
19. In the Application Navigator, navigate to the **images** directory under the **ViewController**, **Web Content** folders.

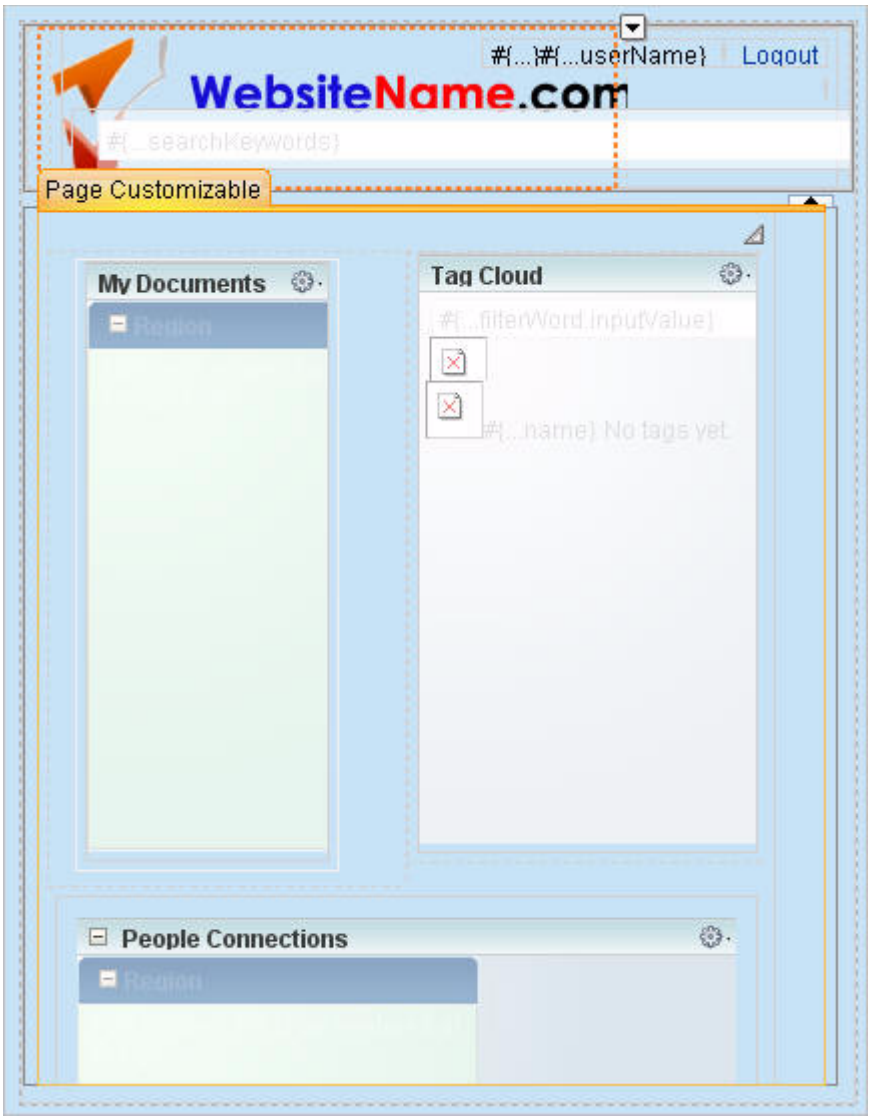
20. Drag and drop `logo2.png` onto the `start` facet and choose **ADF Faces Image** from the context menu (Figure 7–10).

Figure 7–10 Logo2.png in the Structure Window



21. You should now see a visual change to the design time view of your page, as shown in Figure 7–11.

Figure 7–11 MyPage in the Design View with the New Skin



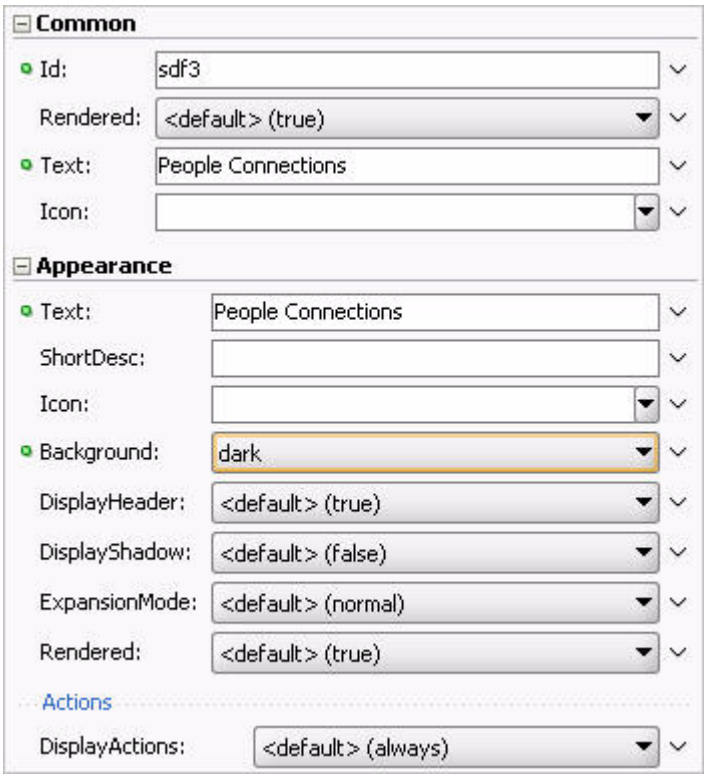
22. Let's make a few changes to the Show Detail Frames surrounding the services. Oracle WebCenter by default allows you to change the appearance of the layout components in your application with three settings: light, medium, and dark. These settings work with your CSS file to slightly alter the appearance of your page. Let's see how this works.

    In the Structure window, select the Show Detail Frame labeled **People Connections** (Figure 7–12).

Figure 7–12 "My Documents" Show Detail Frame in the Structure Window

23. In the Property Inspector for the Show Detail Frame, expand the **Appearance** section.
24. The Background property should currently be set to medium. Use the arrow to change this property to **dark** (Figure 7–13).

Figure 7–13 Setting the Background Property to dark



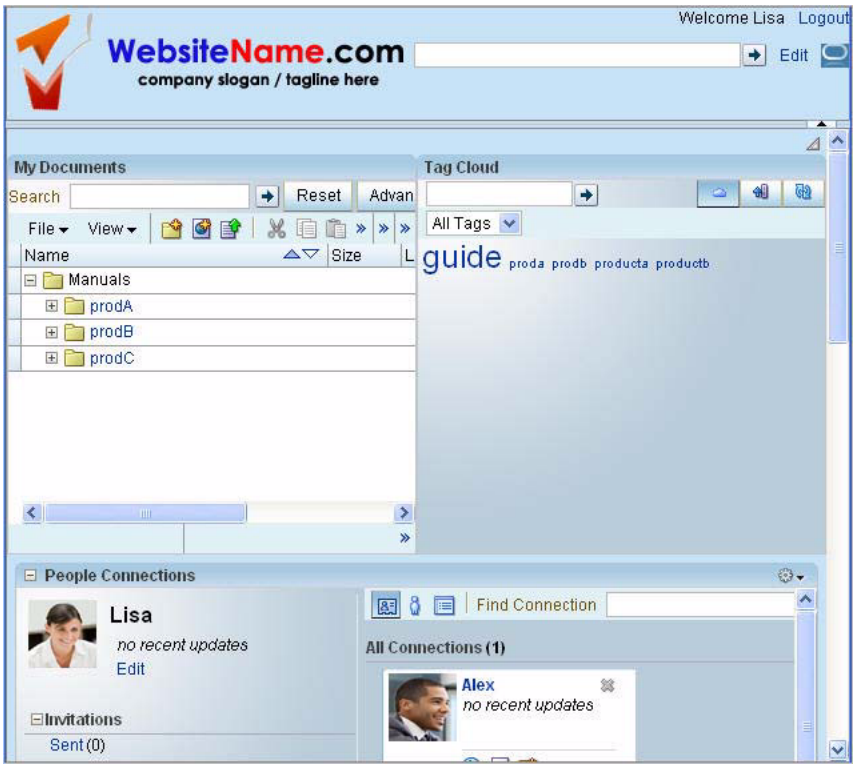
25. Perform the same steps to change the appearance of the Show Detail Frame of all the task flows to **dark**:
    - Activity Stream
    - Messages
    - My Profile
    - My Documents
    - Tag Cloud
    - Mail (optional)
26. In the Structure window, change the same **Background** property to **dark** for the two portlets.

    MyPage should now look like Figure 7–14.

Figure 7–14 Changing the Background of the Show Detail Frames

27. Run the page to your browser to view the new skin and log in as the user `Lisa` with the password `welcome1` (Figure 7–15).

Figure 7–15 Partial View of MyPage at Runtime with the New Skin



Since you are still developing your application (and have not yet deployed it), you can continue to switch back and forth between the runtime view and design time in Oracle JDeveloper to modify the look and feel.

Step 2: Personalize One User's (Lisa's) Page

In Chapter 3, "Creating a WebCenter Application with a Customizable Page," you added Oracle Composer components to your page, including a Change Mode Link. You then worked a little with the Edit mode of the page at runtime to see how you can use Oracle Composer to modify your page at runtime.

Now that you've applied security and added content to the page, you can check out how you and your users can personalize their page (their *own* page) at runtime. For example, when Lisa logs in, she can change the look and feel of her page. When Dan logs into the same application, he does not see Lisa's changes; he only sees his own view. Remember that in Chapter 4, "Adding Security to Your Application," you assigned both the `admin-role` (which is assigned to Lisa) and the `user-role` (which is assigned to both Dan and Alex) privileges to View and Personalize MyPage. This section shows you how these privileges affect what actions a user can perform at runtime.

This step shows you a few tasks you can perform at runtime. You will log in as the three different users: Lisa, Dan, and Alex to see how the views change depending on the user and user privileges. Let's first play with some of the tasks you can perform to personalize Lisa's page.

To personalize Lisa's page:

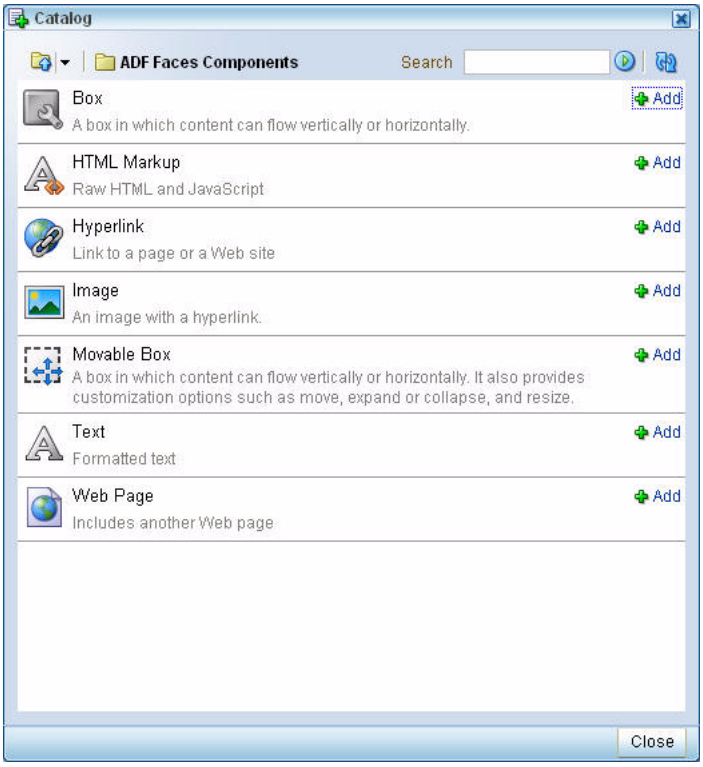
1. If MyPage is not already displaying in your browser, run the page to your browser and log in as `Lisa/welcome1`.

2. One of the tasks Lisa can do at runtime is add a video, for example from YouTube, to her page. In the upper right corner of the page, click **Edit**.
3. Above the People Connections Show Detail Frame, click **Add Content** (Figure 7–16).

Figure 7–16 Add Content Button

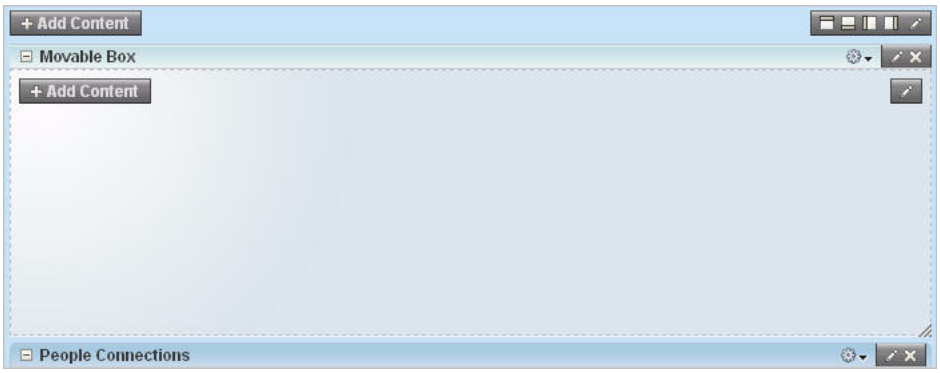
4. In the Catalog that displays, click **ADF Faces Components** to open the folder.
5. Next to **Movable Box,** click **Add** to add this component to the page. This box will contain the video component (Figure 7–17).

Figure 7–17 Adding a Movable Box



6. Click **Close**. The Movable Box displays on your page above the People Connections Show Detail Frame (Figure 7–18).

Figure 7–18 Movable Box on MyPage



7. At the top of the page just below the company logo, notice the Composer toolbar. You'll see a View menu option on the right. Click **View**, then choose **Source** from the list to view the source of the page (Figure 7–19).

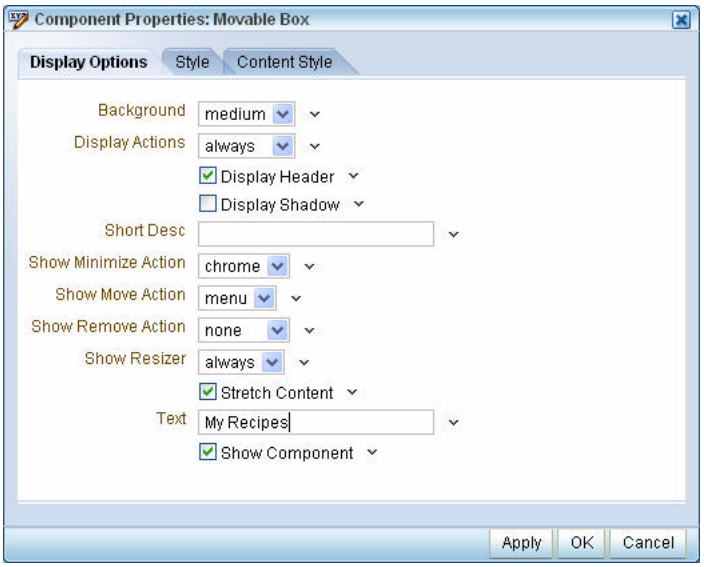
Figure 7–19 Viewing the Source of MyPage at Runtime

8. In the Source View, select the **showDetailFrame:MovableBox**.
9. From the menu, click **Edit** (Figure 7–20).

Figure 7–20 Movable Box in the Source View

10. In the Component Properties: Movable Box dialog, change the **Text** property to `My Recipes`, as shown in Figure 7–21.

Figure 7–21 Setting the Text Property

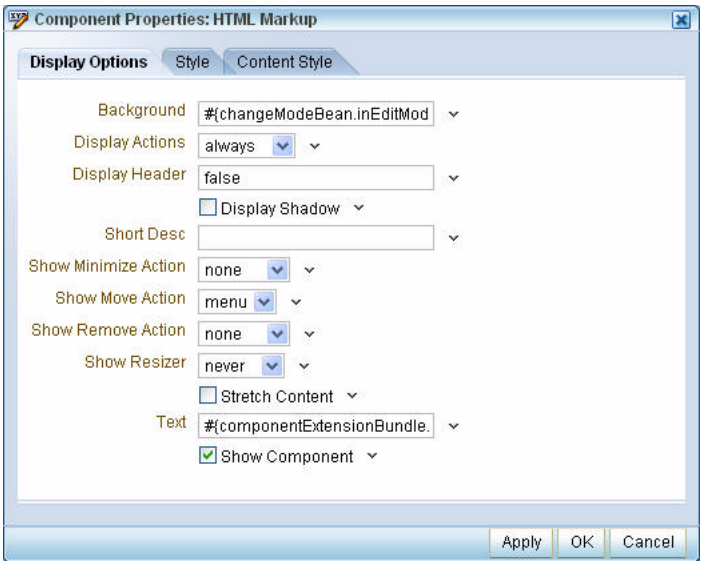


11. Click **Apply**, then click **OK**.
12. Click **Edit** to re-enter the Edit mode of the page.
13. In the My Recipes Show Detail Frame, click **Add Content**.
14. In the Catalog, click **ADF Faces Components**.

Figure 7–22 Add Content Button in the My Recipes Show Detail Frame

15. Next to HTML Markup, click **Add**, then click **Close**.
16. Since the Movable Box Show Detail Frame already contains a header called "My Recipes," you can delete the default header for the HTML Markup Show Detail Frame. Change to the Source view again by clicking **View**, then choosing **Source** in the context menu.
17. Select **showDetailFrame:HTML Markup**, then click **Edit** in the toolbar.
18. In the Component Properties:HTML Markup dialog, set the **Display Header** property to `false` (Figure 7–23).

Figure 7–23 Setting the Display Header Property



19. Click **Apply**, then click **OK**.
20. Close the Source view by clicking **Close** in the upper right corner of the application.
21. Next, add the video component. To do so, click **Edit** again in the upper right corner of MyPage to re-enter Edit mode.
22. In the My Recipes Show Detail Frame, next to the "New HTML Markup" text, hover under the pencil icon to view the toolbar for the component, then click the pencil (Edit) icon to edit the component (Figure 7–24).

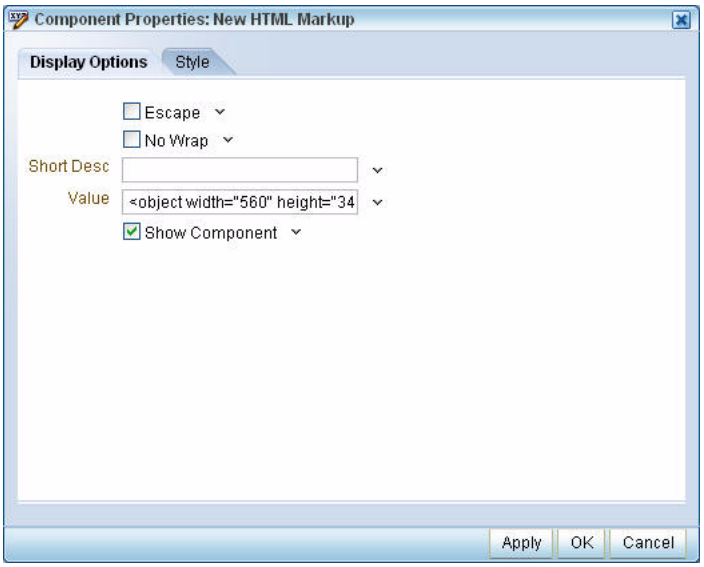
Figure 7–24 Pencil Icon for Editing the HTML Markup Component

23. In the Component Properties:New HTML Markup dialog, change the **Value** property to the following code snippet, as shown in Figure 7–25.

Example 7–2 HTML Markup for the Video Component

```
<object width="560" height="340"><param name="movie"  name="allowFullScreen"
value="http://www.youtube.com/v/8HlqQqP6Mcw&hl=en_US&fs=1&"></param><param
value="true"></param><param name="allowscriptaccess" value="always"></param><embed
src="http://www.youtube.com/v/8HlqQqP6Mcw&hl=en_US&fs=1&"
type="application/x-shockwave-flash" allowscriptaccess="always"
allowfullscreen="true" width="560" height="340"></embed></object>
```

Figure 7–25 HTML Markup in the Value Field



24. Click **Apply**, then click **OK**. You should now see a video component embedded on Lisa's page (Figure 7–26).

Figure 7–26 Video Component on MyPage

25. In addition to adding her favorite video to her page, suppose Lisa wants to change the appearance of her page. Specifically, she wants to see stronger borders around each component.

    While in the Edit mode of the page, click the pencil (Edit) icon in the upper right corner of the My Documents Show Detail Frame.

26. In the Component Properties:My Documents dialog, click the **Style** tab.

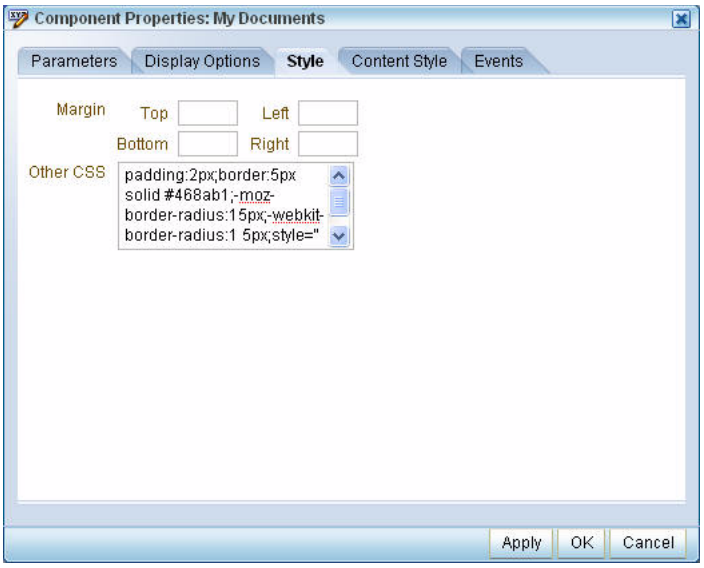
27. On the Style tab, in the **Other CSS** field, enter the following code snippet:

Example 7–3 Changing the Style of the Show Detail Frame

```
padding:2px;border:5px solid
#468ab1;-moz-border-radius:15px;-webkit-border-radius:1 5px;style="
background-color:#ccc;
```

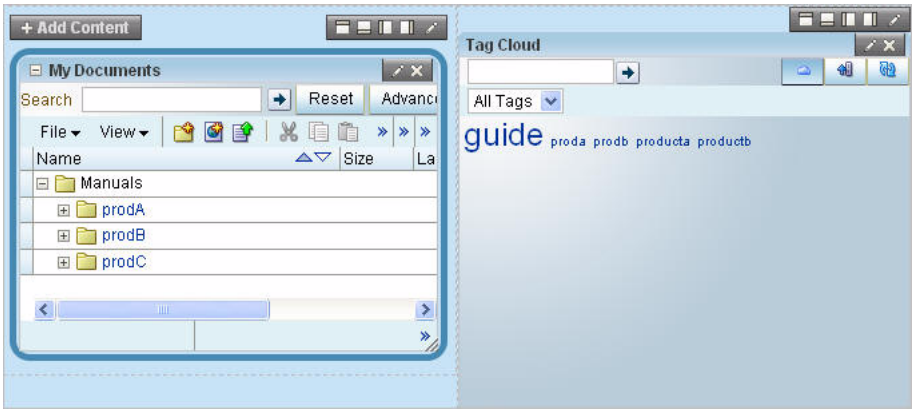
Figure 7–27 shows how the dialog appears after you enter this code.

Figure 7–27 New Border Style in the Component Properties:My Documents Dialog



28. Click **Apply**, then click **OK**. Notice how the border of the My Documents Show Detail Frame changes.

Figure 7–28 New Border on Lisa's Page



29. Now that you've made a few changes to personalize Lisa's page, you can either perform steps 24 through 27 for all the other Show Detail Frames on the page, or continue to the next step ("Step 3: Personalizing a Second User's (Alex's) Page").

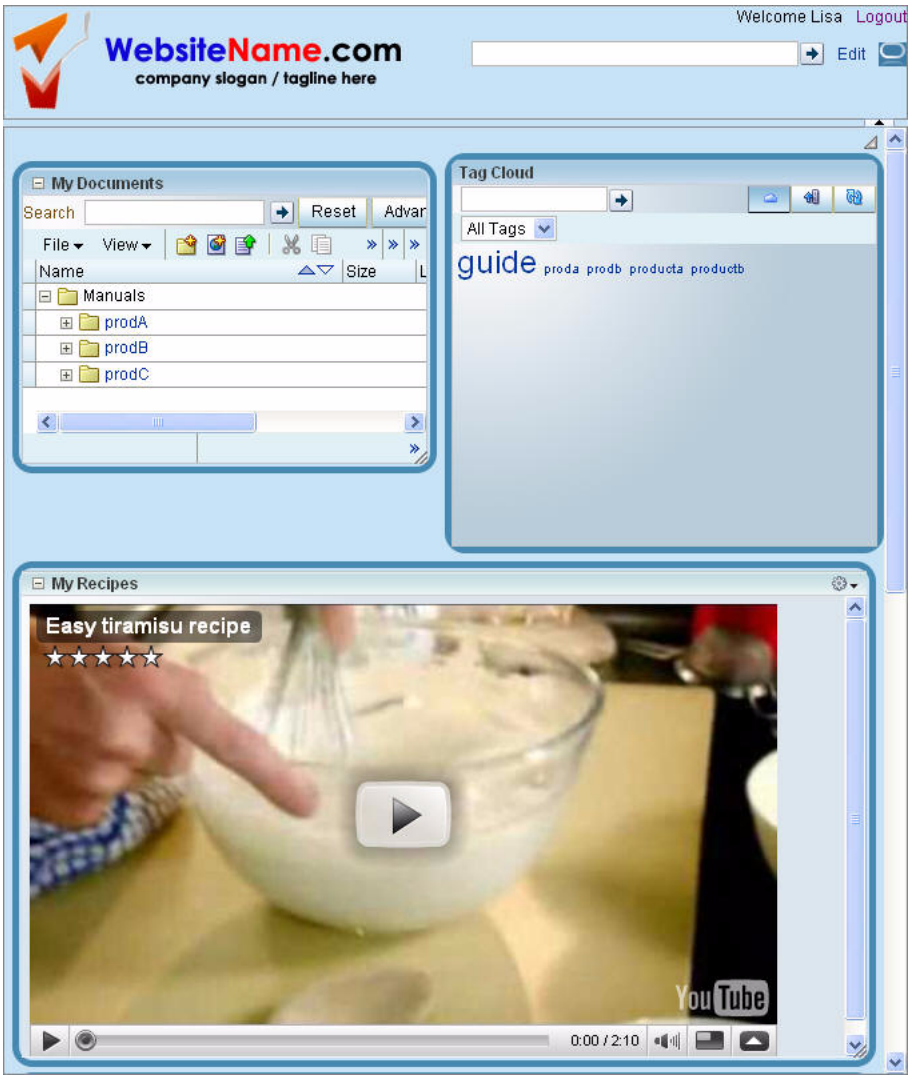
Step 3: Personalizing a Second User's (Alex's) Page

The purpose of switching users at runtime in the application is to show how one user (Lisa) can log in and personalize the application to accommodate her needs (adding the video component and changing the borders), and how these changes do not affect the other users' views of the application. Let's now take a look at Alex's page to see this in practice, as well as make a few changes to the page for Alex.

To personalize Alex's page:

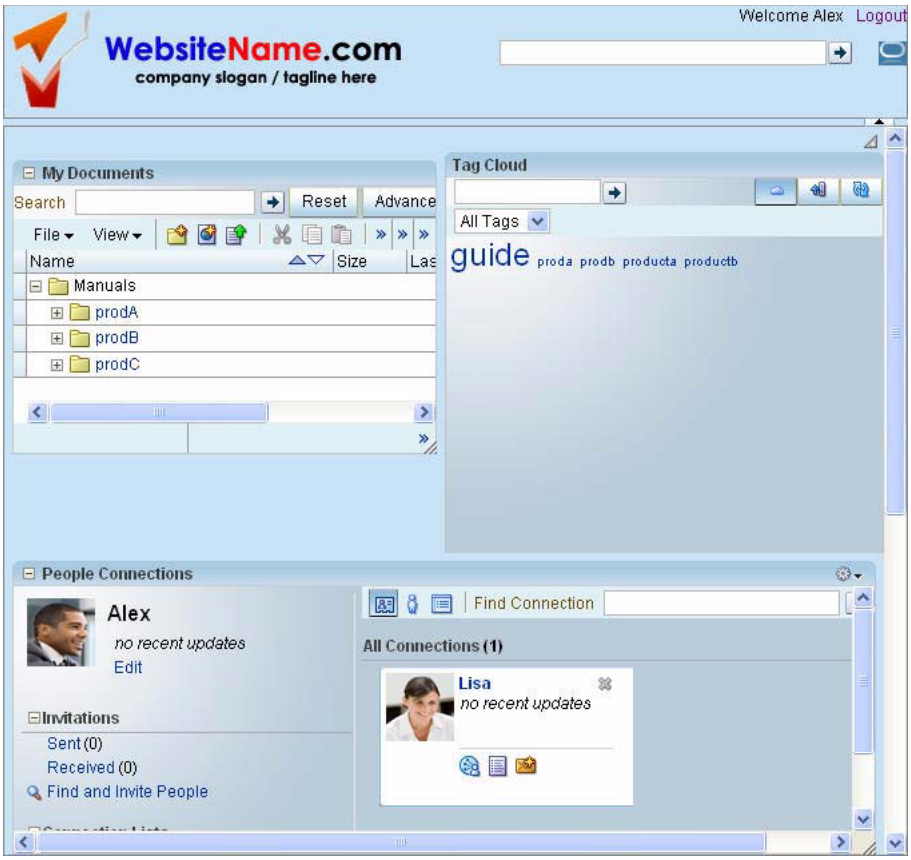
1. While Lisa is still logged into the page, click **Close** to switch from Edit mode to View mode and take a look at Lisa's personalized page (Figure 7–29).

Figure 7–29 Lisa's Personalized Page



2. Click the **Logout** link in the upper right corner of MyPage, then log in as `Alex/welcome1`. Notice that the view of MyPage (Figure 7–30) does not contain any of the changes you made in "Step 2: Personalize One User's (Lisa's) Page".

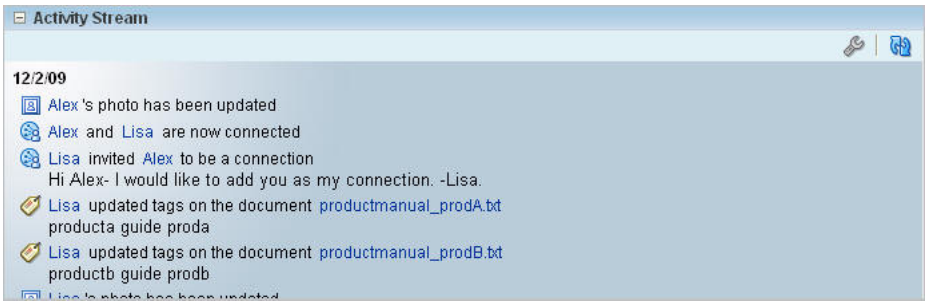
Figure 7–30 Partial View of Alex's Page with No Personalizations



3. Suppose Alex does not want to track updates to the Tag Cloud and Profiles made by other users, but he wants to keep the Activity Stream task flow to track other changes made to the application.

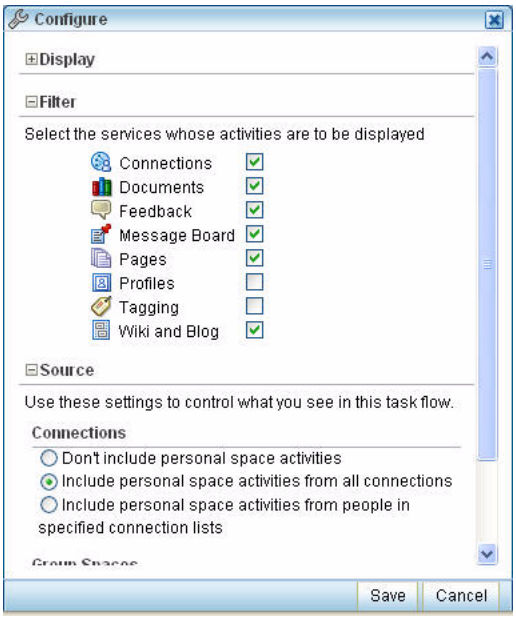
   In the upper right corner of the Activity Stream Show Detail Frame, click the tool (Change the source, filter, and display options) icon (Figure 7–31).

Figure 7–31 Activity Stream Show Detail Frame



4. In the Configure dialog that displays, expand the Filter category, then clear the **Profiles** and **Tagging** checkboxes (Figure 7–32).

Figure 7–32 Configure Dialog



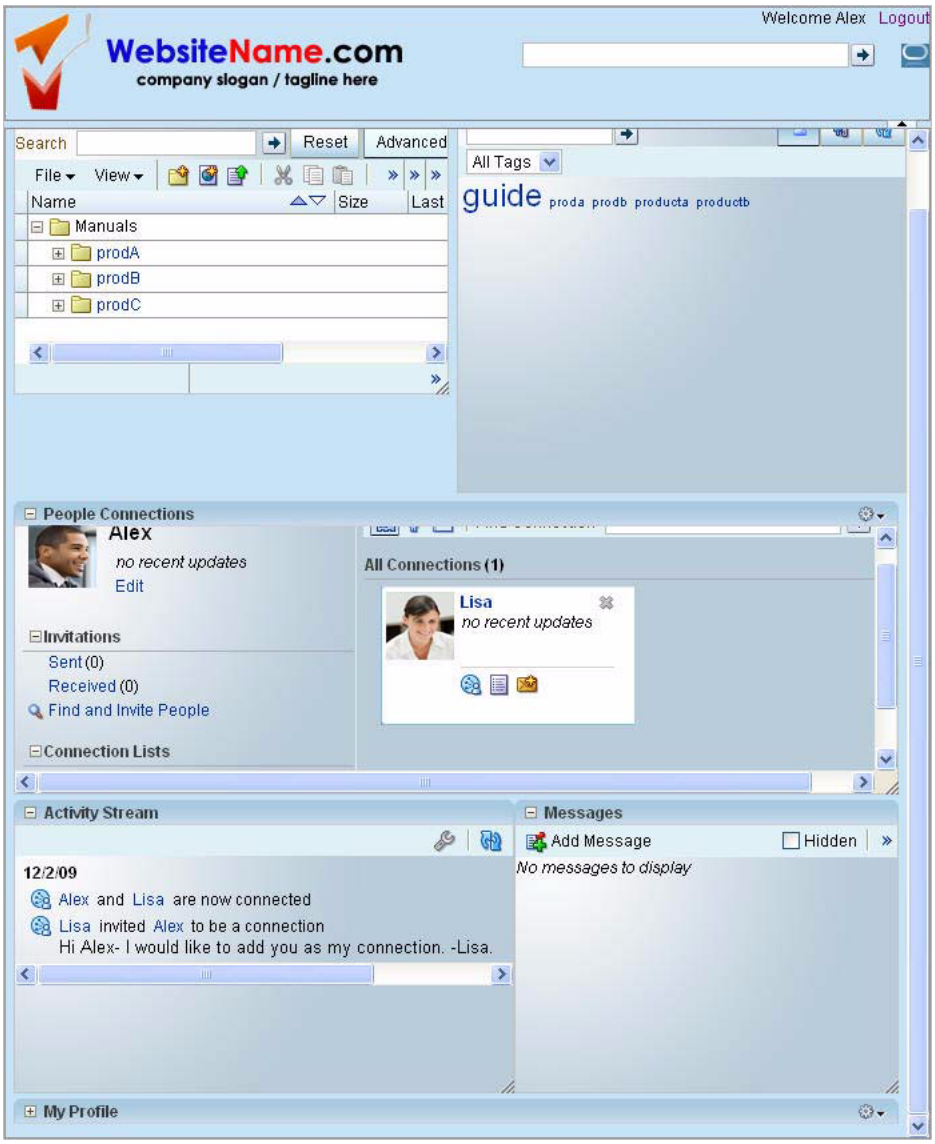
5. Click **Save**. Notice that, to perform this operation, user Alex does not have to be in the Edit mode (which he doesn't have permission to do, anyway). Any user can perform this operation in the View mode; the changes are only visible to the current user.

   Figure 7–33 shows the updated Activity Stream on Alex's page.

Figure 7–33 Alex's Updated View of the Activity Stream

6. While still logged in as user Alex, you can play around with some of the other options available. For example, you can minimize the My Profile Show Detail Frame if you do not wish to see your profile.

Figure 7–34 Alex's Personalized Page



7. In the upper right corner of the page, click the **Logout** link. Now that you are finished personalizing Alex's page, take a look at Dan's page to see what options are available to him in the next step.

Step 4: Personalizing a Third User's (Dan's) Page

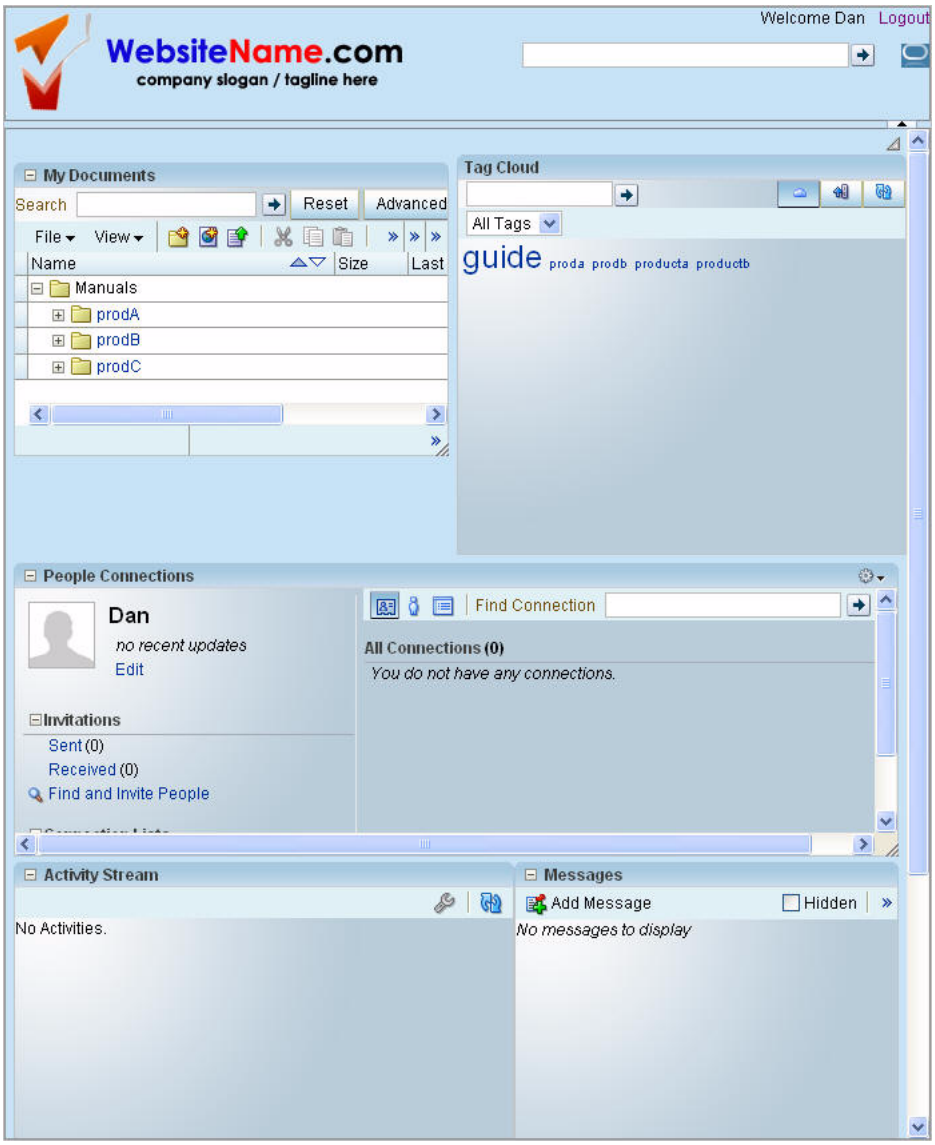
In Chapter 4, "Adding Security to Your Application," you assigned both users Alex and Dan to the `user-role` and gave the role the View and Personalize permissions. Both users, then, can only make minor modifications to the page that only they (while logged in) can see. They do not have the permission to edit the page, and thus cannot perform the same actions as Lisa, who is assigned to the `admin-role` (which can View and Customize the page).

In this step, you will log in as user Dan to verify that he cannot view the changes you made to the application as user Lisa or user Dan. You will also see what changes Dan can make to the application.

To personalize Dan's page:

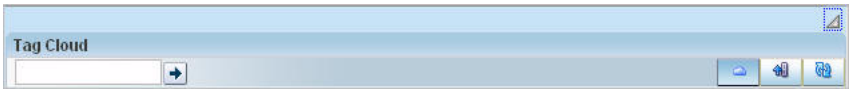
1. Log into the application as `Dan/welcome1`. Notice that none of the changes you made to Lisa's or Alex's page display on Dan's page.

Figure 7–35 Dan's Page



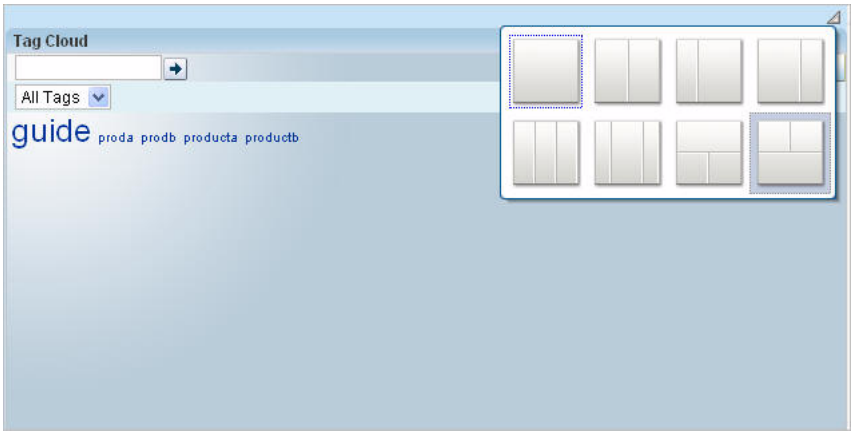
2. While Dan can perform the same actions as Alex, suppose he simply wants to change the layout of the page. Above the Tag Cloud, to the right, notice the change layout icon.

Figure 7–36 Change Layout Icon above the Tag Cloud



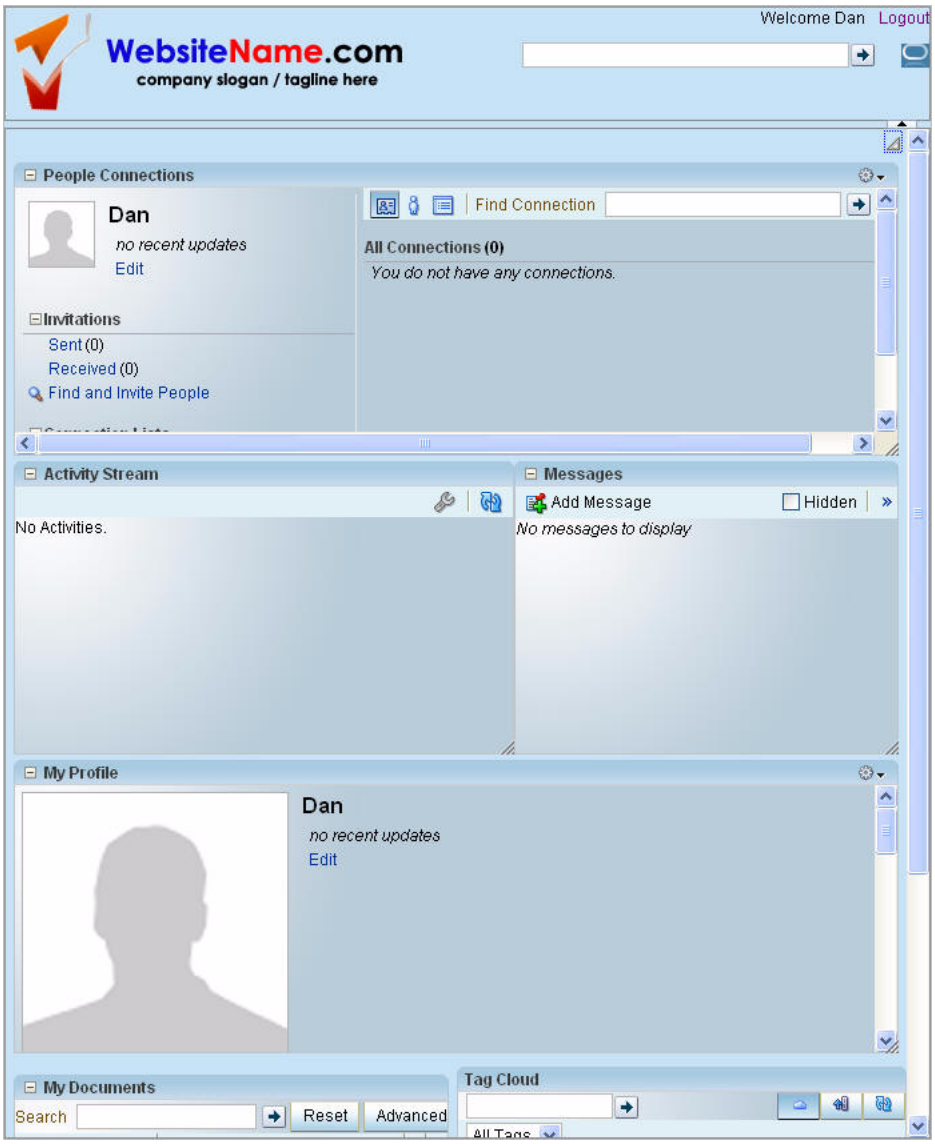
3. Click the icon to view the different layouts you can choose.

Figure 7–37 Choosing a Different Layout



4. Try clicking one of the layouts, such as **Two Columns Below Wide Area** (the layout to the left of the current selection). Notice how the layout changes.

Figure 7–38 New Layout on Dan's Personalized Page



5. As with the changes you made while logged in as users Lisa and Alex, the layout change only displays for Dan. Additionally, Dan does not require Edit privileges to change the layout.

   Click **Logout**.

6. Log in as Lisa or Alex to see that the changes you made in "Step 2: Personalize One User's (Lisa's) Page" and "Step 3: Personalizing a Second User's (Alex's) Page" are still saved, and that Dan's changes do not display for either user Lisa or Alex.

   Now that you have had a short introduction to some of the ways users can personalize their pages, you can learn about more actions you can perform at runtime in Oracle Fusion Middleware User's Guide for Oracle WebCenter.

Congratulations! You've completed this lesson and changed the look and feel of the application by using a skin. You've also checked out how to use Oracle Composer at runtime to learn how to personalize your page at runtime, depending on your user privileges. Continue on to Chapter 8, "Conclusion" to review what you learned in this Tutorial, and where you can find more information about the features you used.

8

Conclusion

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

Summary

In the 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 Services.

Specifically, you learned how to:

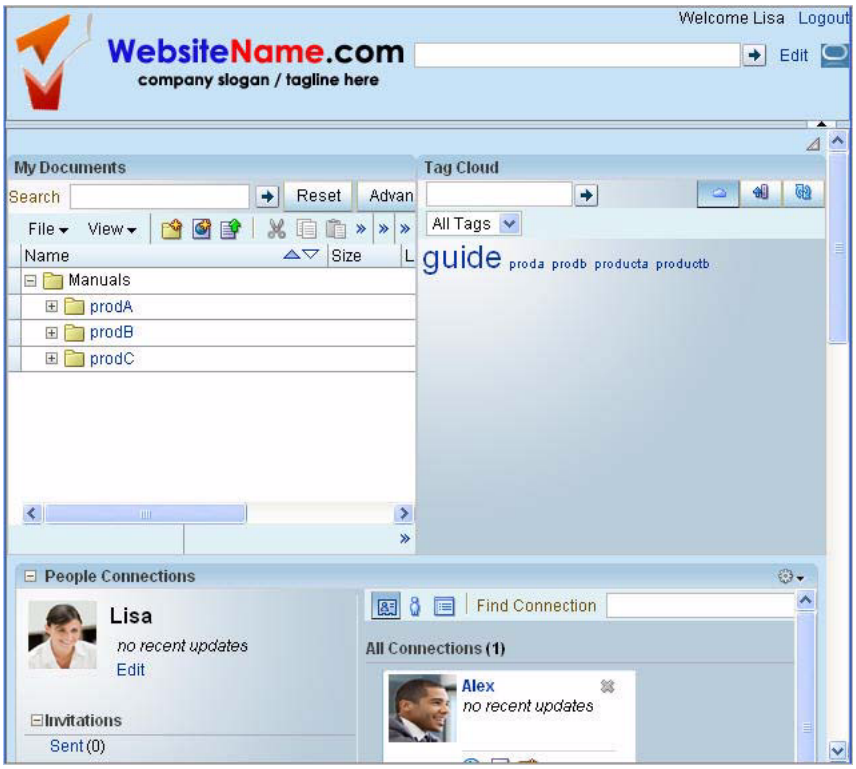
- Create a database connection, which allowed you to access a database containing information your application needed. As you move on and develop more complex custom WebCenter applications, you may want to connect to other databases for various content, and so on. You can use the same methodology to 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 the Tags, People Connections, 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.
- Implement design-time security to a custom WebCenter application to test security and user-related preferences.
- Add and use WebCenter services (Search, Documents, People Connections, Tags, Links, and Mail 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, a "tag cloud" to visualize user-defined tags right in your application, and links to create relationships between content in your application and external sources. You also learned how to create a basic social network, as well as integrate email with the 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).
- Change the look and feel of the application using a skin.
- Personalize your application at runtime while logged in as different users with different permissions. Here, you not only used Oracle Composer to edit your page, you also saw how users with only View and Personalize permissions can modify their page, and how each user only sees the personalization change he or she made while logged in.

Figure 8–1 shows a partial view of the application you created in this Tutorial.

Figure 8–1 Partial View of MyPage at Runtime



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 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

M

O

P

Q

R

S

T

U

V

W