

# **Oracle® WebCenter Framework**

Tutorial

10g (10.1.3.2.0)

**B31072-02**

March 2007

Oracle WebCenter Framework Tutorial, 10g (10.1.3.2.0)

B31072-02

Copyright © 2007, Oracle. All rights reserved.

Primary Authors: Marcie Caccamo, Rosie Harvey

The Programs (which include both the software and documentation) contain proprietary information; they are provided under a license agreement containing restrictions on use and disclosure and are also protected by copyright, patent, and other intellectual and industrial property laws. Reverse engineering, disassembly, or decompilation of the Programs, except to the extent required to obtain interoperability with other independently created software or as specified by law, is prohibited.

The information contained in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. This document is not warranted to be error-free. Except as may be expressly permitted in your license agreement for these Programs, no part of these Programs may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose.

If the Programs are delivered to the United States Government or anyone licensing or using the Programs on behalf of the United States Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the Programs, including documentation and technical data, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement, and, to the extent applicable, the additional rights set forth in FAR 52.227-19, Commercial Computer Software--Restricted Rights (June 1987). Oracle USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

The Programs are not intended for use in any nuclear, aviation, mass transit, medical, or other inherently dangerous applications. It shall be the licensee's responsibility to take all appropriate fail-safe, backup, redundancy and other measures to ensure the safe use of such applications if the Programs are used for such purposes, and we disclaim liability for any damages caused by such use of the Programs.

Oracle, JD Edwards, PeopleSoft, and Siebel are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

The Programs may provide links to Web sites and access to content, products, and services from third parties. Oracle is not responsible for the availability of, or any content provided on, third-party Web sites. You bear all risks associated with the use of such content. If you choose to purchase any products or services from a third party, the relationship is directly between you and the third party. Oracle is not responsible for: (a) the quality of third-party products or services; or (b) fulfilling any of the terms of the agreement with the third party, including delivery of products or services and warranty obligations related to purchased products or services. Oracle is not responsible for any loss or damage of any sort that you may incur from dealing with any third party.

---

---

# Contents

<b>Preface</b> .....	vii
Audience .....	vii
Documentation Accessibility .....	vii
Related Documents .....	viii
Conventions .....	viii
<b>Part I Introducing WebCenter Framework Tutorial</b>	
<b>1 Understanding Oracle WebCenter Suite</b>	
<b>What is Oracle WebCenter Suite?</b> .....	1-1
WebCenter Framework .....	1-3
Building and Consuming Portlets .....	1-3
Customizable Components .....	1-4
Content Integration .....	1-4
Securing Your Application .....	1-4
Managing Your Application Throughout the Lifecycle .....	1-4
WebCenter Services .....	1-5
Oracle JDeveloper .....	1-6
<b>What Will You Learn in this Tutorial?</b> .....	1-6
<b>2 Getting Started</b>	
Downloading Oracle JDeveloper With WebCenter Extensions .....	2-1
Downloading Sample Tutorial Files .....	2-1
Copying the Sample system-jazn-data.xml File .....	2-2
<b>Part II Hands-On Examples</b>	
<b>3 Building and Testing Your First Portlet</b>	
Introduction .....	3-1
Step 1: Using the JSR 168 Java Portlet Wizard .....	3-2
Step 2: Setting Up Your Connection .....	3-10
Step 3: Deploying Your Portlet .....	3-13
Step 4: Creating a JSF Page .....	3-16
Step 5: Registering Your Portlet with Oracle WebCenter Framework .....	3-20

Step 6: Testing the Portlet .....	3-22
Step 7: Adding Some Simple Logic to the Portlet.....	3-26
<b>4 Customizing Your Page</b>	
Introduction.....	4-1
Step 1: Allowing Users To Customize the Page.....	4-2
Step 2: Running and Customizing the Page.....	4-6
Step 3: Making Additional Customizations.....	4-9
Step 4: Testing the New Customizations .....	4-12
Step 5: Changing the Look and Feel .....	4-13
Step 5a: Changing showDetailFrame Background Using the Default ADF Faces Skin .....	4-14
Step 5b: Using Custom Skins to Apply Your Own Styles.....	4-15
<b>5 Adding the Rich Text Portlet</b>	
Introduction.....	5-1
Prerequisites.....	5-2
Step 1: Registering the Rich Text Producer .....	5-2
Step 2: Adding the Rich Text Portlet To Your Page.....	5-3
Step 3: Customizing the Rich Text Portlet at Run Time.....	5-5
<b>6 Making Portlets Communicate</b>	
Introduction.....	6-1
Prerequisites.....	6-2
Step 1: Registering Portlet Producers .....	6-2
Step 2: Placing the Parameter Form Portlet on a Page.....	6-5
Step 3: Customizing the Parameter Form Portlet .....	6-9
Step 4: Placing an OmniPortlet on a Page .....	6-10
Step 5: Building an OmniPortlet That Uses a Web Service.....	6-12
Step 6: Configuring the Portlets Together .....	6-15
Step 7: Testing Portlet Interaction .....	6-16
<b>7 Adding Content to Your Page</b>	
Introduction.....	7-1
Prerequisites.....	7-2
Step 1: Creating a Data Control .....	7-2
Step 2: Adding Content to a Page at Design Time .....	7-5
Step 3: Displaying Folder Content in a Table .....	7-8
Step 4: Displaying Folder Content in a Tree .....	7-17
Step 5: Searching Folder Content.....	7-26
<b>8 Providing Security</b>	
Introduction.....	8-1
Prerequisites.....	8-2
Step 1: Creating a Login Page.....	8-2
Step 2: Configuring ADF Security Settings .....	8-5

Step 3: Creating a Welcome Page.....	8-10
Step 4: Securing Pages .....	8-22
Step 5: Mapping Security Roles in orion-web.xml.....	8-25
Step 6: Demonstrating the Security Features .....	8-30
Login as User Singh .....	8-30
Login as User Cho .....	8-32
Login as User Harvey .....	8-33
Login as User King.....	8-34
Try to Access a Secured Page Directly .....	8-34
Enter Invalid Credentials .....	8-35
Step 7: Authorizing Access to Your Data Controls .....	8-35

## 9 Deploying Your WebCenter Application

Introduction.....	9-1
Prerequisites .....	9-1
Step 1: Creating a WebCenter Application Deployment Profile .....	9-2
Step 2: Deploying Directly to the Preconfigured OC4J .....	9-5
Step 3: Migrating Your Security Policy .....	9-6
Step 4: Running Your Deployed Application .....	9-7
Step 5: Using Application Server Control Console to Manage WebCenter Applications .....	9-8
Summary .....	9-10

## Part III Appendixes

### A How to Set Up the Tutorial Identity Store

Creating Users .....	A-2
Creating Roles and Assigning User Members.....	A-4
Making Tutorial Users and Roles Available to JDeveloper's Authorization Editor .....	A-6

## Index



---

---

# Preface

This tutorial introduces you to Oracle WebCenter Framework, one of the key components of Oracle WebCenter Suite. 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 WebCenter Framework Developer's Guide* for assistance.

---

---

**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 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) (Purpose and basic architecture)
- Oracle ADF Faces
- Java

The tutorial is intended for the developer who wants to build a 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, with good usability, to the disabled community. 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.

### **TTY Access to Oracle Support Services**

Oracle provides dedicated Text Telephone (TTY) access to Oracle Support Services within the United States of America 24 hours a day, seven days a week. For TTY support, call 800.446.2398.

## **Related Documents**

For more information on Oracle WebCenter Framework, see the following documents:

- *Oracle WebCenter Framework Developer's Guide*
- *Oracle WebCenter Framework Error Messages Guide*

For more information on Application Development Framework, see the following documents, both of which are available on Oracle Technology Network (OTN) at <http://www.oracle.com/technology/index.html>:

- *Oracle Application Development Framework: Tutorial*
- *Oracle Application Development Framework: Developer's Guide*

## **Conventions**

The following text conventions are used in this document:

<b>Convention</b>	<b>Meaning</b>
<b>boldface</b>	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
<i>italic</i>	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
monospace	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.



# Part I

---

---

## Introducing WebCenter Framework Tutorial

Part I contains the following chapters:

- [Chapter 1, "Understanding Oracle WebCenter Suite"](#)
- [Chapter 2, "Getting Started"](#)



---

---

# Understanding Oracle WebCenter Suite

This chapter introduces you to Oracle WebCenter Suite and helps you understand how you can use it to enhance your service-oriented applications. With Oracle WebCenter Suite, you get services that you can integrate with your application to afford your users improved communication, content management capabilities, customization, and advanced search support. More important, you get a development framework that provides essential capabilities, such as the ability to consume portlets and content in a Java Server Faces application, declarative security, and life-cycle management tools.

In this chapter, you will discover answers to these key questions:

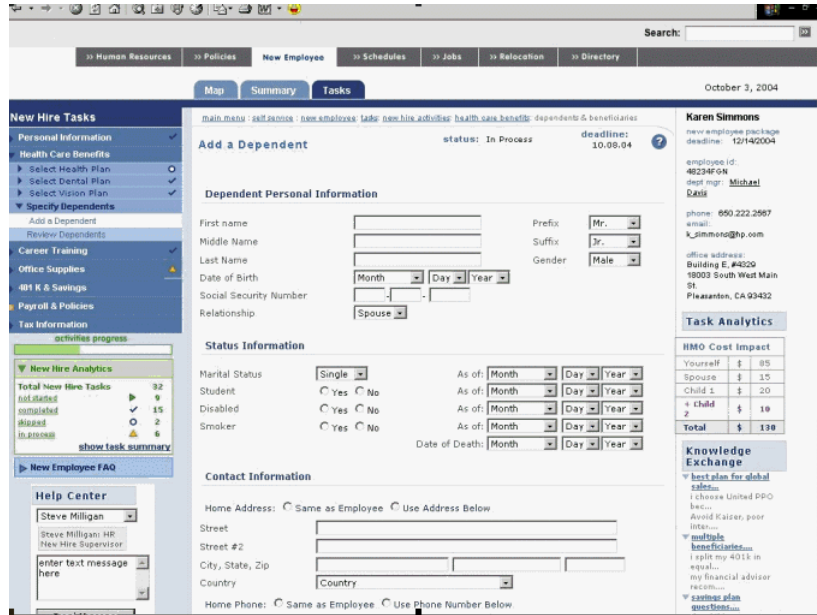
- [What is Oracle WebCenter Suite?](#)
- [What Will You Learn in this Tutorial?](#)

After you read this chapter, you'll be ready to start building your own Java EE 5 application.

## What is Oracle WebCenter Suite?

As key technologies like Wiki, RSS, and blogs change the landscape of the Internet by empowering individuals across the globe, user demand for applications that simplify transactions becomes more pronounced. One way to simplify transactions is to provide everything the user needs to support a given task within the application itself. Consider the example shown in [Figure 1-1](#):

**Figure 1–1 Sample Application**



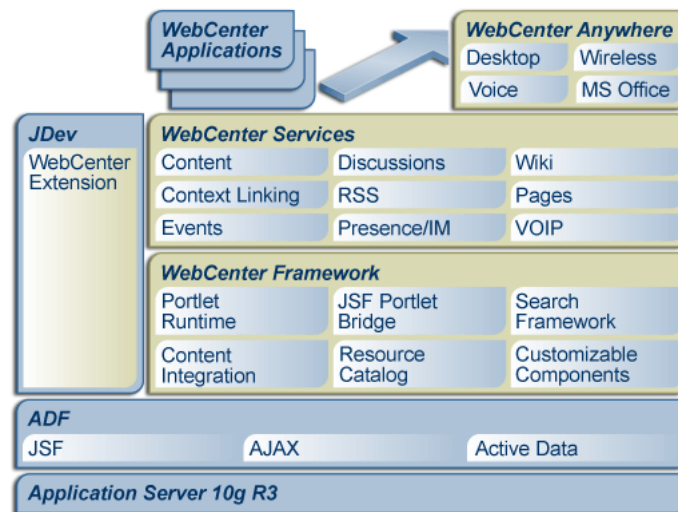
In this example, a user who is new to the company is working with an application that allows him to add dependents to his company insurance policy. Notice that the transaction itself is surrounded by additional context that helps the user, including:

- New Hires Tasks, in the upper left corner, provide an activity guide that shows where the user is in the larger process of becoming acclimated to his new company. The user's next task is also identified. This type of process orchestration helps the user step through the entire multi-step flow quickly and easily.
- Task and process analytics let users know where they are in the process and how decisions are impacting them. In this case, the Task Analytics on the right show the total cost impact of the benefit choices made so far.
- The Help Center on the bottom left provides an up to date FAQ for quick access to typical questions and a direct chat link to the help center where the user can ask additional questions not addressed by the FAQ. Again, no need for the user to leave the context of the transaction to get help.
- Knowledge Exchange, on the bottom right, provides documentation relevant to the current task. These documents, stored in the corporate repository, give detailed advice on the different beneficiary and dependent scenarios applicable to the user.

Until Oracle WebCenter Suite, building this kind of application was a rather tedious process. To gain access to the beneficiary scenarios, for example, used to involve creating a portlet to gain a view into the Java Content Repository (JSR 170)—if the API required to do so was available. Oracle WebCenter Suite reduces the front-end labor historically required to bring necessary business components to the user by capitalizing on the notion of Service Oriented Architecture (SOA). Thanks to Oracle WebCenter Suite's commitment to SOA, as well as to the Java Content Repository (JSR 170) and other industry standards, you get a wide range of plug-and-play products, tools, and services that make it easy to build the applications your users need.

Figure 1–2 shows what Oracle WebCenter Suite provides<sup>1</sup>:

<sup>1</sup> Some components shown are not available in the initial release of Oracle WebCenter Suite: Presence/IM, Discussions, and Wiki. In this chapter, we describe components relevant to this release of Oracle WebCenter Framework.

**Figure 1–2 Oracle WebCenter Suite**

Let's examine these building blocks in more detail.

## WebCenter Framework

WebCenter Framework augments the Java Server Faces (JSF) environment by providing additional integration and runtime customization options. In essence, it integrates capabilities historically included in portal products directly into the "fabric" of the JSF environment. This eliminates artificial barriers for the user and provides the foundation for developing the kinds of context-rich applications depicted in [Figure 1–1](#).

### Building and Consuming Portlets

Portlets help you bring data from the Web, database, and so on, into your application. Using Oracle JDeveloper, you can create your own standards-based portlets to be consumed by any JSR 168 or WSRP-compatible portal. The Oracle Application Server Portal Developer Kit (PDK) has been enhanced to support extended portlet capabilities as defined by WSRP 2.0 within the structure of the Java Portlet Standards APIs. From a WebCenter application, you can consume JSR 168, WSRP 1.0, WSRP 2.0 or Oracle PDK-Java portlets all within the same application, or even within the same page.

Several pre-built portlets are available for use through a preconfigured OC4J that is automatically available to you through JDeveloper. Two such portlets, OmniPortlet and Web Clipping, help empower users to gather their own data, while the Rich Text portlet enables users to publish their own announcements and bulletins. You can make these portlets available to users by dropping them on your page, or you can use them yourself to create the specific portlets your users will need.

- **OmniPortlet:** A portlet that enables users to easily publish data from a variety of sources, using a variety of layouts. Users can base an OmniPortlet on almost any kind of data source, such as spreadsheets (character-separated values), XML, Web Services, and even application data from an existing Web page. Once the data has been obtained, they can format it using layouts such as bulleted lists, charts, HTML, and so on.

As a developer, you might want to use this tool to gather and format the data for your users—for example, to create an employee directory—then place it on your

page for user consumption. Once you do so, the portlet becomes available through JDeveloper's Component Palette for others to use in their applications.

- **Web Clipping:** An extremely easy-to-use wizard that requires no technical expertise whatsoever. Users simply locate the Web content they want to "clip", then use the wizard to grab it and display it within their application. If the Web content on the original site is updated, so is the user's associated Web Clipping.
- **Rich Text portlet:** A tool that allows users to publish their own announcements and broadcasts. When you place a Rich Text portlet on a page, during run time, authorized users can access all the rich-text editing tools needed to insert, update, and format display text.

### **Customizable Components**

WebCenter Framework provides new JSF components that allow developers to make any of their applications customizable. These new components act as containers into which developers can drop another Faces view component or a portlet. With these capabilities in place, administrators can customize virtually any JSF page by minimizing/maximizing, hiding/showing, or moving any component on the page.

### **Content Integration**

Suppose you have data in a content management system such as Oracle Content DB, OracleAS Portal—or even on your file system—that you want to make available to your application. WebCenter Framework provides you with the JCR adapters you need to access that content. Using JDeveloper, you can then build JCR data controls to grab the content and drop it onto your page in a variety of display modes. WebCenter Framework also comes with Oracle Drive, through which you can represent the contents of your OracleAS Portal repository as a tree-like structure, right on your desktop.

### **Securing Your Application**

With the ADF extensions provided in WebCenter Framework, you can define security for an entire application, a page within the application, or for individual actions provided by customizable components. In this tutorial, you will learn how to create a simple login page and assign basic privileges to several distinct users.

In many cases, it is desirable to leverage existing applications that have their own authentication mechanism, such as e-mail. WebCenter Framework provides the means to embed those applications through the use of the External Application wizard. See the *Oracle WebCenter Framework Developer's Guide* for more information.

### **Managing Your Application Throughout the Lifecycle**

WebCenter Framework reduces the time required to build, deploy, and migrate your applications through the use of several tools:

- **Development framework:** Oracle JDeveloper and ADF provide the tools and framework you need to build and update your application. Adding portlets, content, and customization capabilities to your WebCenter application is simply a matter of dragging and dropping the appropriate objects in either a source or WYSIWYG environment. To simplify the test and debug phases, WebCenter Framework includes a deployment profile (WebCenter Application WAR) that packages and migrates your portlet customizations, content, and page customizations to any J2EE container (such as the standalone OC4J provided) so you can test and debug your application before deploying it to a production server.

- Enterprise deployment: When you're ready to deploy your application to a production environment, Oracle WebCenter Framework's Predeployment Tool packages and migrates your portlet customizations to your production location, changes the pointer to your content repository, and ensures that the application points to your production Metadata Services location. When the Predeployment Tool completes its work, you get a target EAR file that you can then deploy to the final location using Enterprise Manager.
- Standards-based administration: Browser-based tools allow administrators to deploy, configure, and manage WebCenter applications. In addition, tools built on industry standards-based JMX methods offer administrators granular control and monitoring mechanisms for health status, performance, and popularity. Tools for obtaining historical performance and status reporting over time (within a single Oracle Application Server context) are also provided. WebCenter application metrics are delivered using the familiar Application Server Control monitoring and management interface.

## WebCenter Services

WebCenter Services offer a variety of content management, search, and communication services, including:

- Oracle Content Database (Oracle Content DB), the default content repository for Oracle WebCenter Services. Oracle Content DB is a full-fledged content management system that enables users to manage content through the Web or from desktop applications. A rich library of ready-to-use Web services is provided to content-enable your enterprise in a service-oriented environment. With Oracle Content DB, you can:
  - Improve the productivity of individuals and teams with secure access to the right content in the context of business processes
  - Reduce risk associated with content, including information loss and legal discovery
  - Facilitate adaptability of business processes
  - Reduce IT and administrative costs through content consolidation

Oracle Content DB bridges the gap between limited capability file servers and the specialized, expensive, and complex content management applications that are so widely available.

- Oracle Secure Enterprise Search is a crawler-based service that can search a multitude of sources, structured and unstructured, in a variety of file formats, indexed or real-time. With Oracle Secure Enterprise Search, you can reduce the time spent finding relevant documents on your company's information repositories.
- Communication Services, which help you better connect people and facilitate communication. These services include:
  - Instant Messaging: Lets users freely exchange ideas through audio and video feeds, file exchange, and a range of other capabilities.
  - Presence Server: *Presence* provides information about a person's availability to every person or application that subscribes to that person's status. Chats and other real-time services can be initiated from the associated user interface.
  - Discussion forum: An interactive message board for sharing information, questions, and comments.

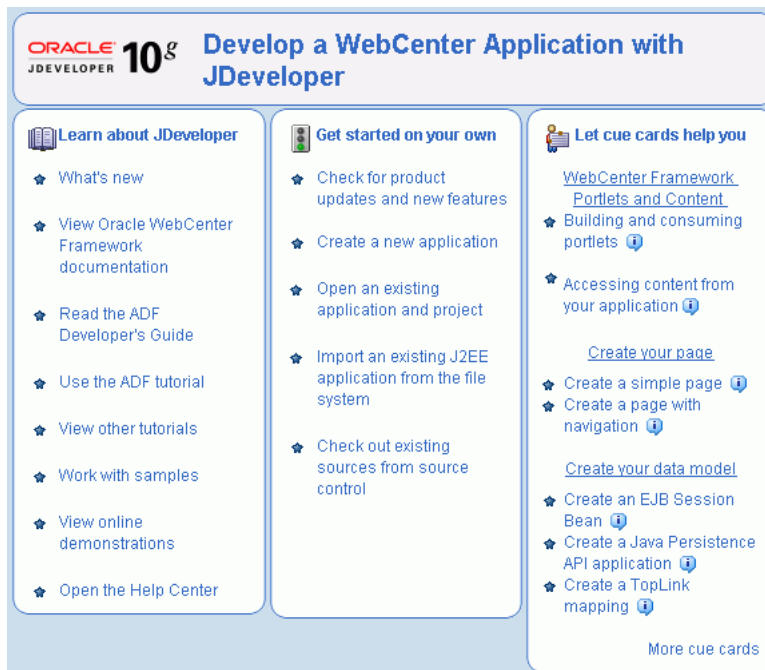
- Wiki is server software that allows users to freely edit and create Web page content using a Web browser. This ease of interaction and operation makes Wiki an effective tool for collaborative communication.

## Oracle JDeveloper

Oracle JDeveloper is an integrated development environment (IDE) for building service oriented applications using the latest industry standards for Java, XML, Web services, and SQL. Oracle JDeveloper supports the complete software development life cycle, with integrated features for modeling, coding, debugging, testing, profiling, tuning, and deploying applications. Oracle JDeveloper's visual and declarative approach and the Oracle Application Development Framework(ADF) work together to simplify application development and to reduce mundane coding tasks. For example, code for many standard user interface widgets, such as buttons, list of values, and navigation bars, are pre-packaged for you. All you have to do is select the appropriate widget from the Component Palette and drop it into your application.

As you work through this tutorial, you will become more familiar with Oracle JDeveloper and the advantages it offers. For more information about Oracle JDeveloper, access one of the many educational aids from the Oracle JDeveloper Start Page (Figure 1-3), accessible from Oracle JDeveloper's Help menu.

Figure 1-3 Oracle JDeveloper Start Page



## What Will You Learn in this Tutorial?

In this tutorial, you will use JDeveloper to build an application containing five simple pages:

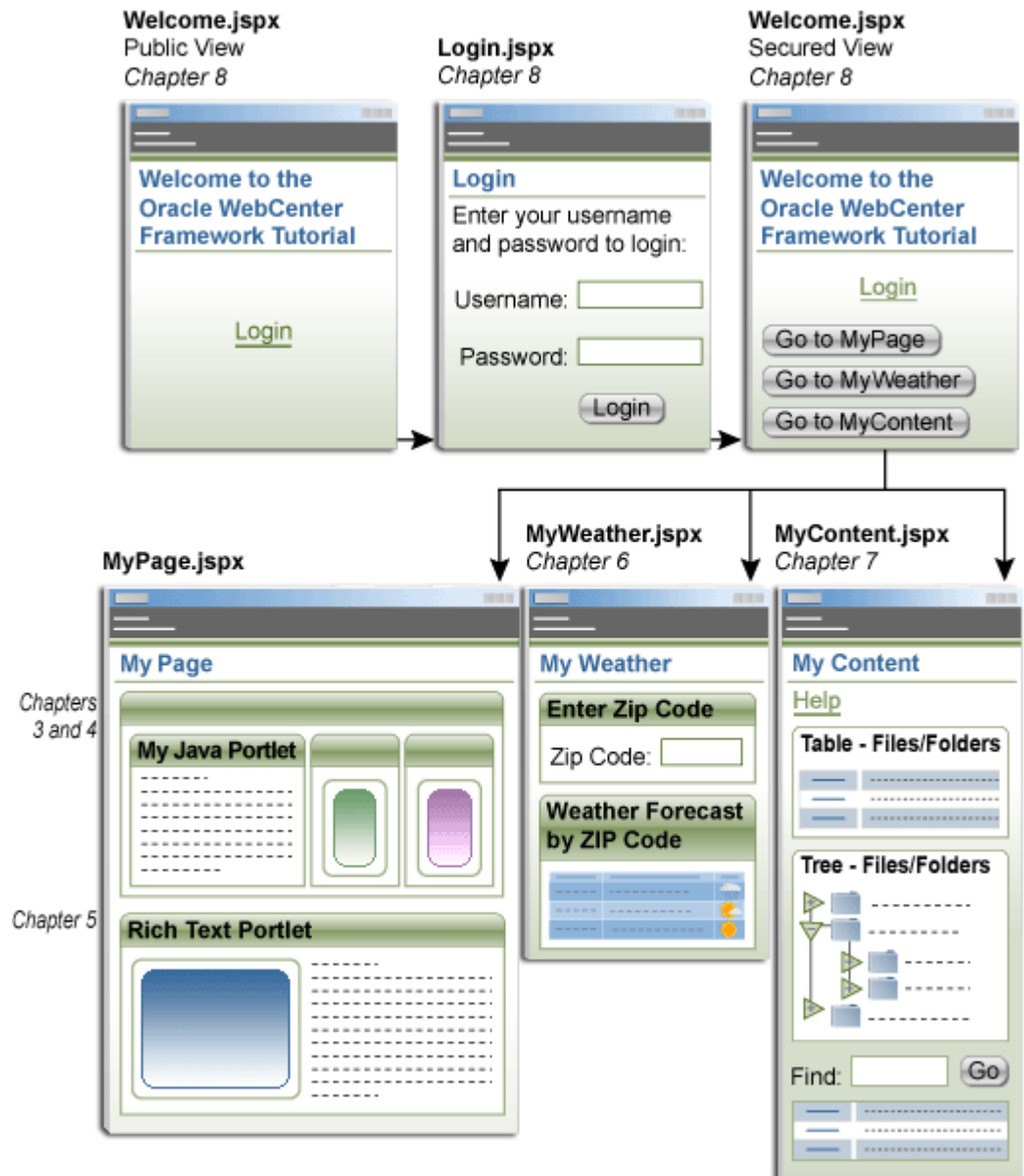
- A Welcome page, which displays public content to public users and secure content to authenticated users.
- A Login page, through which you will learn the basics of how to allow users to authenticate themselves.



- MyPage, upon which you will place a Java portlet, the Rich Text portlet, and images from your file system.
- MyContent, upon which you will publish content from your file system.
- MyWeather, through which you will learn how to use OmniPortlet and the Parameter Form Portlet, and how to enable communication between the two.

The logical flow between the pages is illustrated in the following graphic (Figure 1–4):

**Figure 1–4 Flow Between Pages**



Note that you will not develop these pages in the order they are presented in the graphic. Rather, you will start with a very basic page, MyPage, then move gradually from there into more complex topics.

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, "Getting Started"](#) tells you what you need to do before you can complete the steps in this tutorial. Be sure to complete all the steps described in this chapter.
- [Chapter 3, "Building and Testing Your First Portlet"](#) shows you how to build a basic page, MyPage, add a simple portlet, then enhance the portlet to embrace more sophisticated logic.
- [Chapter 4, "Customizing Your Page"](#) introduces you to the means by which you enable customization for your application. In this lesson, you'll continue to work with MyPage.
- [Chapter 5, "Adding the Rich Text Portlet"](#) shows you how to place the Rich Text portlet on MyPage.
- In [Chapter 6, "Making Portlets Communicate"](#), you'll create a new page, MyWeather, and add a Parameter Form Portlet and OmniPortlet to it. You'll also learn how to set up parameters between the two portlets to enable simple communication.
- [Chapter 7, "Adding Content to Your Page"](#) shows you how to add content from your file system onto a page called MyContent. You'll also learn how to add a search form to the page.
- In [Chapter 8, "Providing Security"](#), you'll learn how to implement security by creating a Welcome page and a Login page. Only privileged users will have access to MyPage, MyWeather, MyContent, and secure content on the Welcome page.
- [Chapter 9, "Deploying Your WebCenter Application"](#) will show you the steps involved in deploying your sample application.

If you've never used JDeveloper before, by the end of this tutorial you should have a fairly solid grasp of the fundamental purpose and capability of the tool. Of course, JDeveloper, as well as the framework it rests upon, ADF, both offer tremendous powers that are only briefly explored in this tutorial. You will no doubt want to learn more about both products before you begin developing in earnest. Here are two excellent resources, both available on Oracle Technology Network, <http://www.oracle.com/technology/>:

- [Application Development Framework Tutorial](#)
- [Application Development Framework Developer's Guide](#)

Let's get started!

---

---

## Getting Started

This chapter tells you how to download the correct version of Oracle JDeveloper and some files you'll need to complete the lessons in this tutorial.

### Downloading Oracle JDeveloper With WebCenter Extensions

To complete this tutorial you'll need access to or have installed Oracle JDeveloper Studio Edition (10.1.3.2.0 or later). This release includes all the functionality required to build WebCenter applications.

You can download Oracle JDeveloper Studio Edition (10.1.3.2.0) from the Oracle Technology Network at:

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

Unzip to a directory of your choice, which in this tutorial is referred to as *JDEVHOME*.

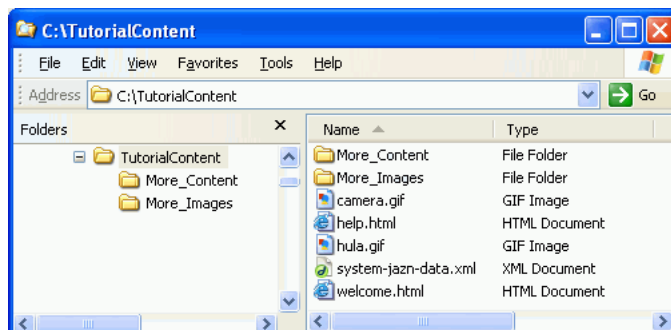
### Downloading Sample Tutorial Files

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:

1. Open a browser, and enter the following in the Address field:  
<http://www.oracle.com/technology/products/webcenter/files/webcentertutorialcontent.zip>
2. To open the zip file (`webcentertutorialcontent.zip`), click **Open**.
3. Unzip the file to a local drive, such as C.

[Figure 2-1](#) shows the file unzipped to: `C:\TutorialContent`

**Figure 2-1** Sample Content ZIP File Unzipped



4. Make a note of the location where the sample content is now stored, as you'll need to access it occasionally as you work through the tutorial.

Next, set up some user data required to complete [Chapter 8, "Providing Security"](#).

## Copying the Sample system-jazn-data.xml File

The tutorial download (`webcentertutorialcontent.zip`) includes a sample `system-jazn-data.xml` file. This file contains the user data you'll need to complete [Chapter 8, "Providing Security"](#). To use the sample data, you'll need to copy the sample `system-jazn-data.xml` file to several Oracle JDeveloper locations. These instructions tell you where to copy the file, and we advise you to back up the original files before doing so.

We provide this file for your convenience and to simplify the tutorial. Normally, when building applications, you would not overwrite an existing `system-jazn-data.xml` as you may lose users, roles, and policies you've added through JDeveloper.

---

---

**Caution:** If you are already building secure applications with JDeveloper and have populated `system-jazn-data.xml` with user data of your own, you might prefer to define the tutorial users/roles from scratch so they co-exist alongside your current user data. If this is the case, do not *copy* the sample `system-jazn-data.xml` file as described here. Instead, follow all the instructions in [Appendix A, "How to Set Up the Tutorial Identity Store"](#) before starting [Chapter 8, "Providing Security"](#).

---

---

To copy the sample `system-jazn-data.xml` file:

1. Locate the sample `system-jazn-data.xml` file in the directory where you unzipped the sample tutorial content. For example: `C:\TutorialContent`
2. Back up `system-jazn-data.xml` files at the following locations before making any modifications for the purposes of this tutorial:
  - **Oracle JDeveloper's Embedded OC4J -**  
`JDEVHOME\jdev\system\oracle.j2ee.10.1.3.xx.xx\embedded-oc4j\config\system-jazn-data.xml`  
  
The system directory is created when you open JDeveloper for the first time. If you do not see a system folder in your `JDEVHOME\jdev` directory, launch JDeveloper and shut it down again before performing this step.
  - **Oracle JDeveloper -**  
`JDEVHOME\j2ee\home\config\system-jazn-data.xml`
3. Copy the sample `system-jazn-data.xml` file to the following directories:
  - `JDEVHOME\jdev\system\oracle.j2ee.10.1.3.xx.xx\embedded-oc4j\config`  
  
When you copy this file you're making the user data available to JDeveloper's embedded OC4J.
  - `JDEVHOME\j2ee\home\config`  
  
When you copy this file you're making the user data available to the Oracle JDeveloper Authorization Editor. More about this in [Chapter 8](#).

You're now ready to move to your first lesson, [Step 1: Using the JSR 168 Java Portlet Wizard in Chapter 3, "Building and Testing Your First Portlet"](#).



# Part II

---

## Hands-On Examples

Part II contains the following lessons:

- [Chapter 3, "Building and Testing Your First Portlet"](#)
- [Chapter 4, "Customizing Your Page"](#)
- [Chapter 5, "Adding the Rich Text Portlet"](#)
- [Chapter 6, "Making Portlets Communicate"](#)
- [Chapter 7, "Adding Content to Your Page"](#)
- [Chapter 8, "Providing Security"](#)
- [Chapter 9, "Deploying Your WebCenter Application"](#)





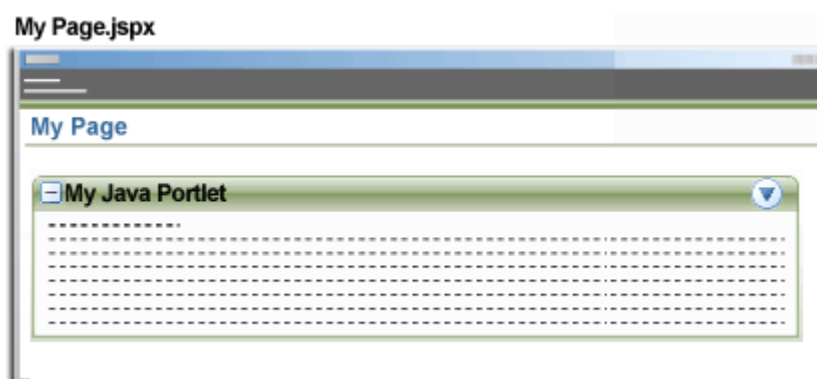
---

## Building and Testing Your First Portlet

In this lesson you will learn how to build your first portlet. After you create the portlet, you will test it, create a simple page, drop the portlet on the page, test it again, then add some additional logic to the portlet. By the end of this lesson, you should have a good handle on what's involved with building and testing a simple portlet.

Figure 3–1 shows a conceptual picture of what you'll have at the end of this lesson: a WebCenter application named MySample containing a page called MyPage (MyPage.jspx). You'll also create a portlet called MyJavaPortlet, which you'll place upon MyPage and then customize.

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



### Introduction

We will build a portlet and test it in the following steps:

- [Step 1: Using the JSR 168 Java Portlet Wizard](#)
- [Step 2: Setting Up Your Connection](#)
- [Step 3: Deploying Your Portlet](#)
- [Step 4: Creating a JSF Page](#)
- [Step 5: Registering Your Portlet with Oracle WebCenter Framework](#)
- [Step 6: Testing the Portlet](#)
- [Step 7: Adding Some Simple Logic to the Portlet](#)

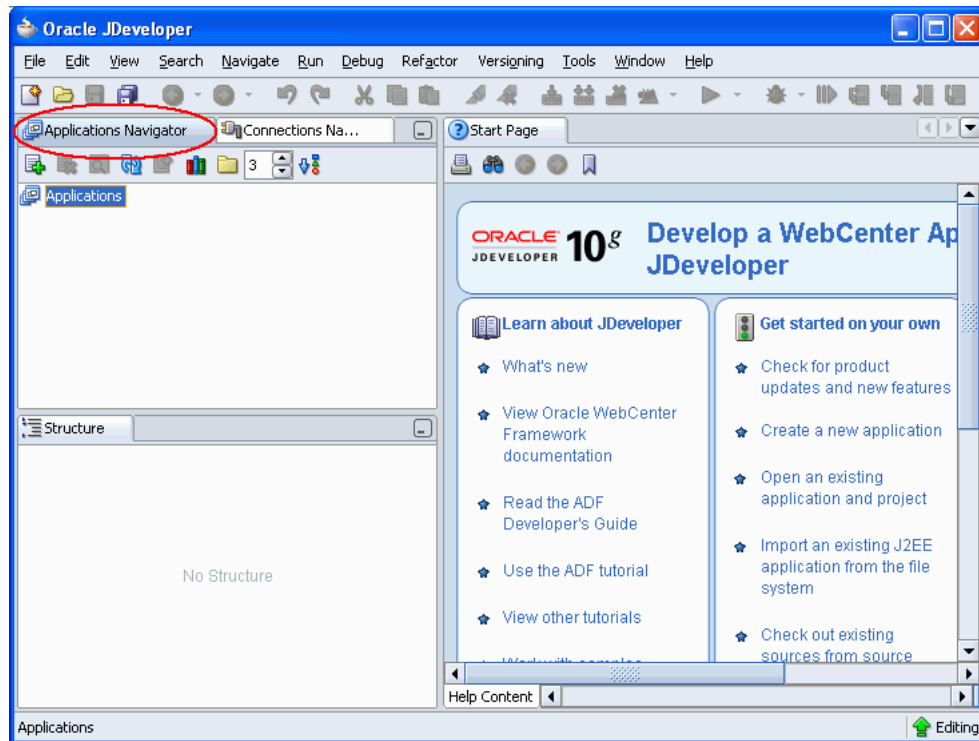
The process will be roughly the same no matter what type of portlet you build using JDeveloper.

## Step 1: Using the JSR 168 Java Portlet Wizard

Before you create a portlet, you must create an application to serve as a container for your portlet. To create an application:

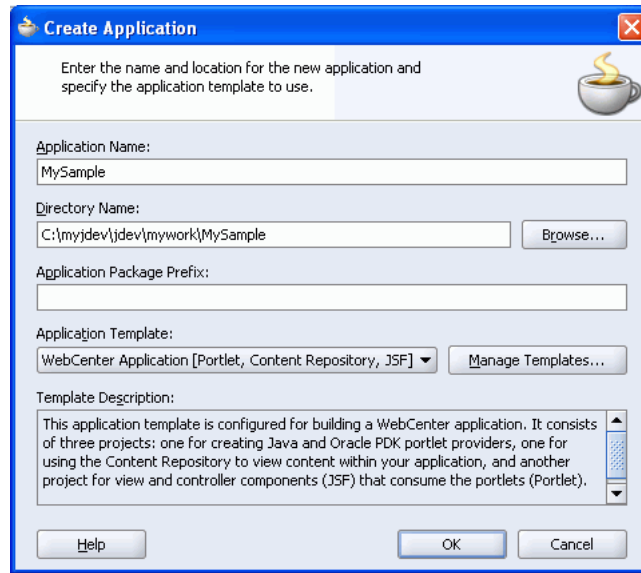
1. Start up JDeveloper by double clicking `jdev.exe` wherever JDeveloper resides.
2. If a Tip of the Day appears, click **Close** to close it.
3. Ensure that the Applications Navigator tab is highlighted (Figure 3–2).

**Figure 3–2 Applications Navigator in JDeveloper**



4. In the Applications Navigator, highlight **Applications**, and right-click.
5. Select **New Application**.
6. In the Application Name field, enter: `MySample`
7. Accept the default location for `MySample`, and make a note of it. You'll need it later when you get to [Step 3: Deploying Your Portlet](#).
8. From the Application Template pull-down as shown in [Figure 3–3](#), choose **WebCenter Application**.

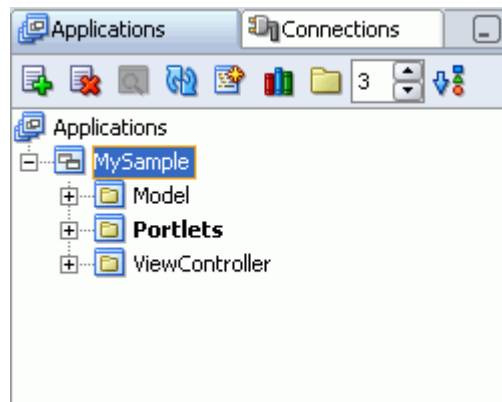
This template creates the projects that you need, and helps target JDeveloper so that only those options applicable to a WebCenter application are presented to you.

**Figure 3–3** *Creating a New Application*

9. Click **OK**.

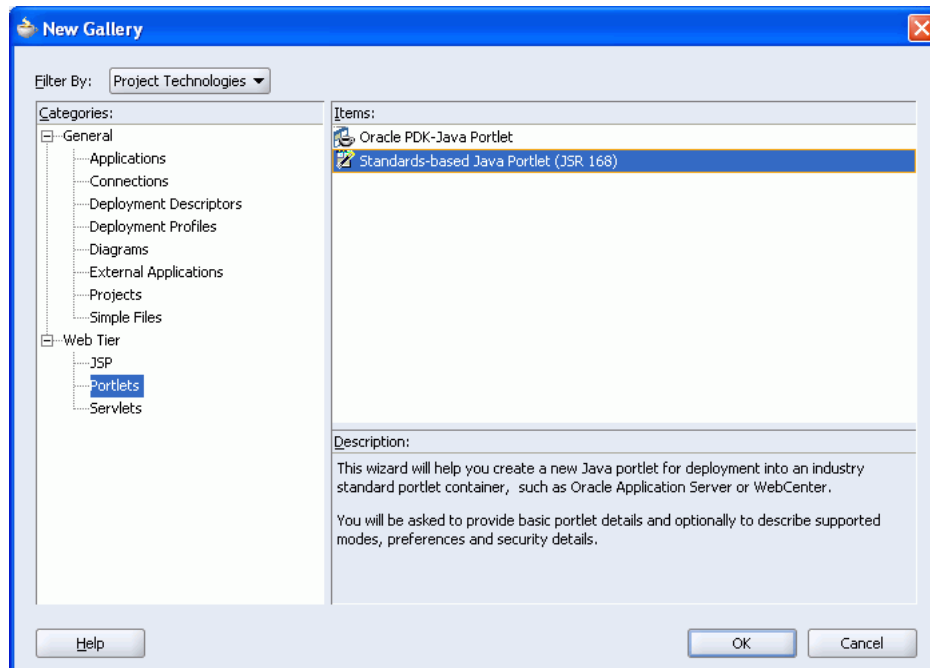
In the Applications Navigator as shown in [Figure 3–4](#), you can see that a WebCenter application consists of three projects:

- **Model**, in which you define the JavaBeans and other data controls you need if the application is to perform any back-end logic.
- **Portlets**, in which you'll create your portlet.
- **ViewController**, in which you'll create the JavaServer Faces page that will consume the portlet.

**Figure 3–4** *The New MySample Application*

Now let's invoke the JSR 168 Java Portlet Wizard to create your portlet.

10. In the Applications Navigator, right-click **Portlets**, and select **New**.
11. In the Categories pane as shown in [Figure 3–5](#), expand the **Web Tier** category, and click **Portlets**.

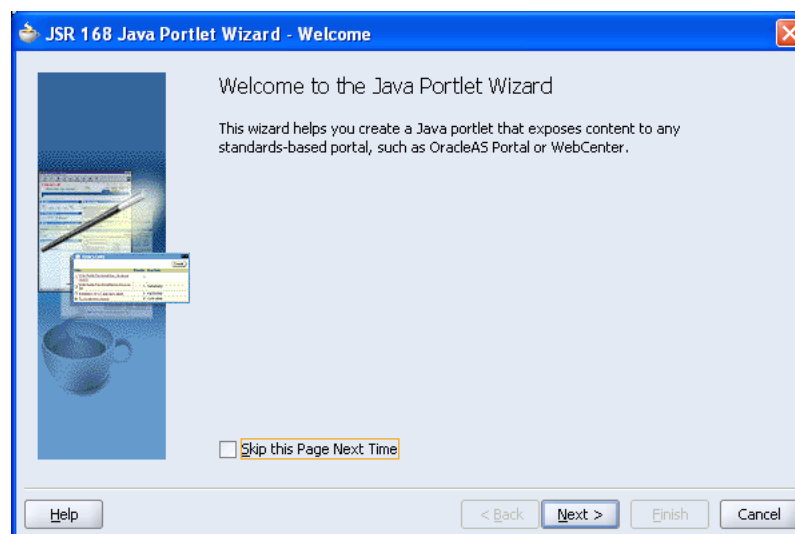
**Figure 3–5 New Gallery for Portlets**

Notice there are two kinds of portlets you can create:

- An Oracle PDK Java portlet. Select this option when the portlet you want to build will be consumed by WebCenter applications, OracleAS Portal, or some other type of Oracle-specific solution. You build an Oracle PDK Java portlet using the APIs provided by the PDK.
- A standards-based (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.

**12. Highlight **Standards-based Java Portlet (JSR 168)**, and click **OK**.**

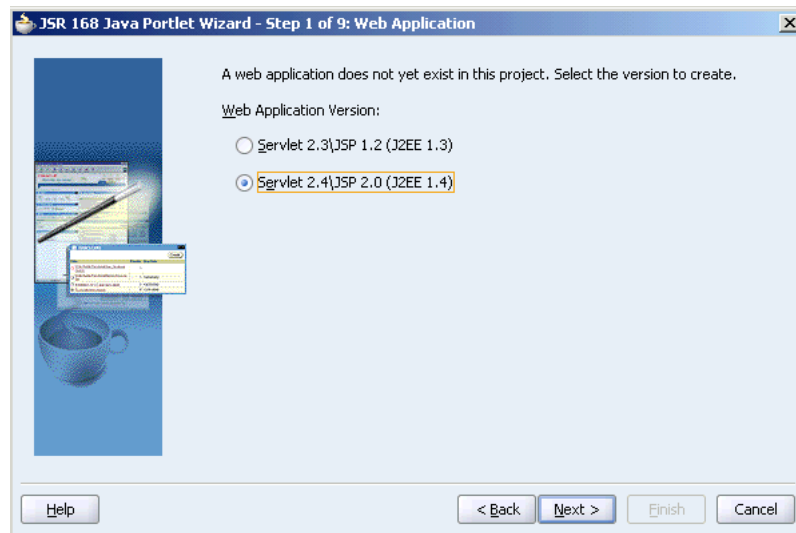
This opens the JSR 168 Java Portlet Wizard as shown in [Figure 3–6](#).

**Figure 3–6 JSR 168 Java Portlet Wizard Welcome Page**

The JSR 168 Java Portlet Wizard generates a skeleton for the portlet, to which you add your own logic. Let's see how this is done.

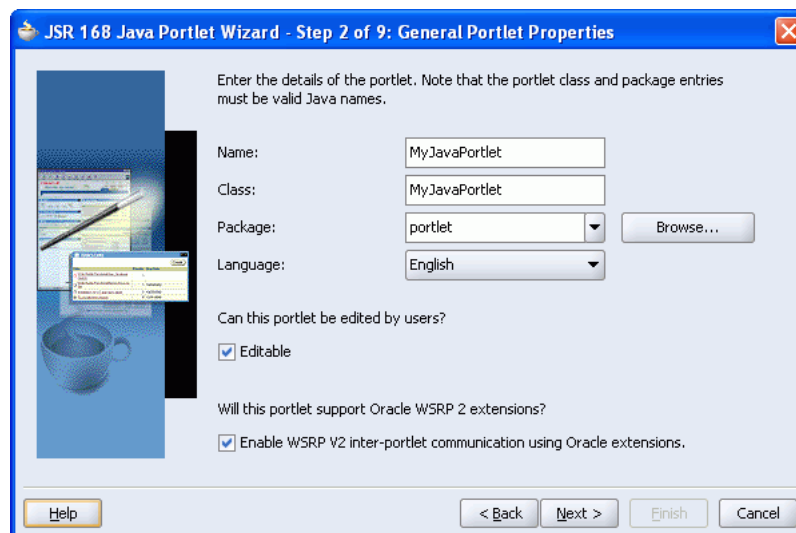
13. Click **Next** to move beyond the Welcome page.
14. Leave the default specified on the Web Application page (Figure 3-7). The servlet version makes certain tag libraries available. Because they are backward compatible, it's always best to select the latest version, unless you have a specific reason for selecting an older version.

**Figure 3-7** *Selecting a Web Application Version*



15. Click **Next**.
  16. On the General Portlet Properties page, change the Name and Class names to `MyJavaPortlet` (no spaces) as shown in Figure 3-8.
- It's usually a good idea to make the class name the same as the display name, so the portlet that displays on the page has the same name as what appears in the Applications Navigator.

**Figure 3-8** *General Portlet Properties Page*



**17. Select *Enable WSRP V2 inter-portlet communication using Oracle extensions*.**

You'll work with WSRP inter-portlet communication later in this tutorial.

**18. Accept the other defaults that appear on the General Portlet Properties page (Figure 3-8).**

Make sure the **Editable** box is checked, as you'll want to allow users to personalize this portlet during runtime. (The Editable box makes the portlet personalizable. A user *personalizes* a portlet when he or she makes changes that are unique to that user. A user with a higher set of permissions can *customize* the portlet to make changes that everyone sees. More about this later.)

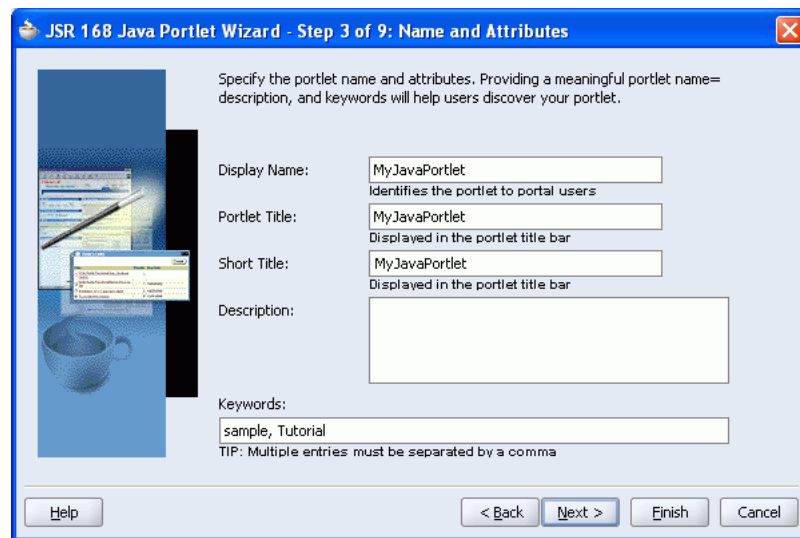
**19. Click *Next*.**

**20. On the Name and Attributes page, enter the values shown in Figure 3-9.**

**Table 3-1 Name and Attribution Values**

Property	Value
Display Name	Name that will appear in the JDeveloper Component Palette. Because you entered <b>MyJavaPortlet</b> as the class name, this field is already populated with that name.
Portlet Title	Title that will appear on the portlet header. Because you entered <b>MyJavaPortlet</b> as the class name, this field is already populated with that name.
Short Title	Title that will appear on the portlet header on mobile devices. You can leave this field populated, although we won't use it.
Description	Description of the portlet. This field is relevant only when the portlet is used in an OracleAS Portal 10g environment. Leave this field blank.
Keywords	Enter <b>sample, Tutorial</b> . 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 OracleAS Portal 10g, they are supported by other vendors from whom you may have obtained a deployment environment.

**Figure 3-9 Name and Attributes Page**



**21. Click *Next*.**

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.

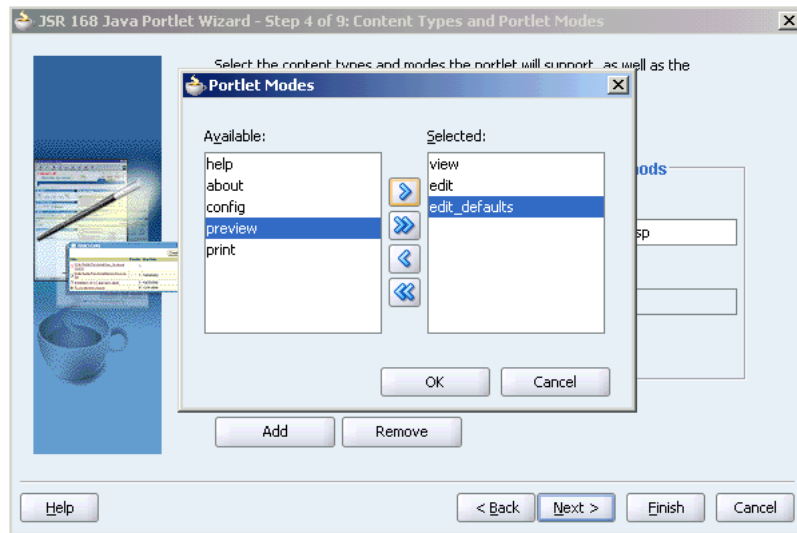
Later on in this tutorial we'll want to test the difference between Personalize and Customize using this portlet. That means we need to enable customization for the portlet now. We do that by adding the `edit_defaults` content type to the portlet's configuration.

22. In the Content Types and Portlet Modes pane as shown in [Figure 3-10](#), highlight **view**, and click **Add**.

The Portlet Modes window is displayed.

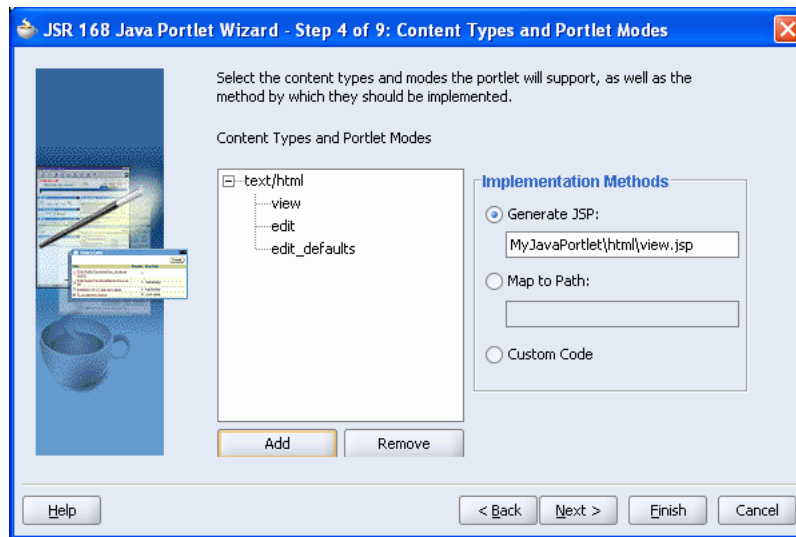
23. Highlight **edit\_defaults**, and use the arrow to move it to the **Selected** pane.

**Figure 3-10 Adding a Portlet Mode**



24. Click **OK**.

Before we leave the Content Types and Portlet Modes page, take a look at the **Implementation Methods** area as shown in [Figure 3-11](#). These controls enable you to specify whether you want to generate JSP for the portlet, or if you want to use your own custom JSP code. In this lesson, we'll ask JDeveloper to generate JSPs for us.

**Figure 3–11** Selecting an Implementation Method

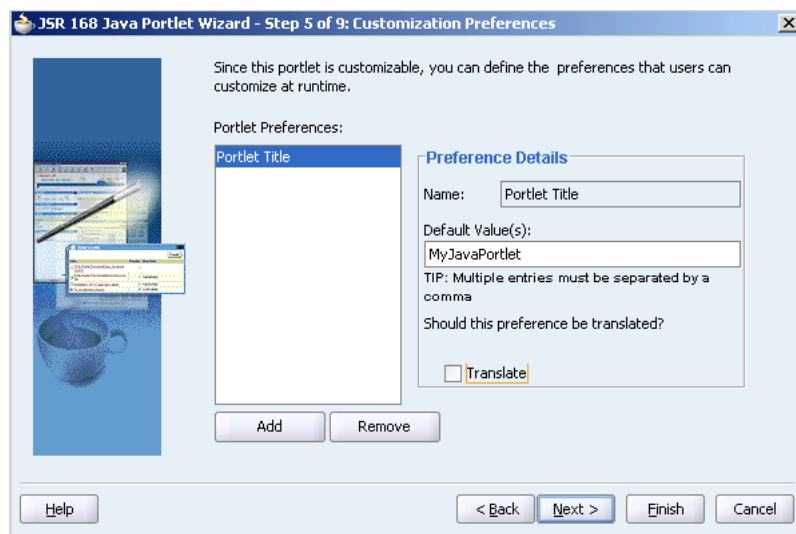
25. Click **Next**.

Although you could click **Finish** here and produce a basic portlet, let's continue working through the wizard so you can see what other options are available.

26. On the Customization Preferences page as shown in [Figure 3–12](#), enter `MyJavaPortlet` in the Default Values field.

Because we specified that we want this portlet to be **Editable**, users will be able to change this title at runtime.

We're not interested in translating for this release, so de-select the **Translate** check box.

**Figure 3–12** Setting Default Values

We're not going to do anything with this page now, but 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.

27. Click **Next**.

28. On the Security Roles page, click **Next**.

This page is used to specify which of the application's security roles you want to establish for this portlet.

29. On the Caching page, click **Next**.

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

30. On the Initialization Parameters page, click **Next**.

Initialization parameters provide the Web application developer a way to configure the behavior of the portlet. For this tutorial, no initialization parameters are needed.

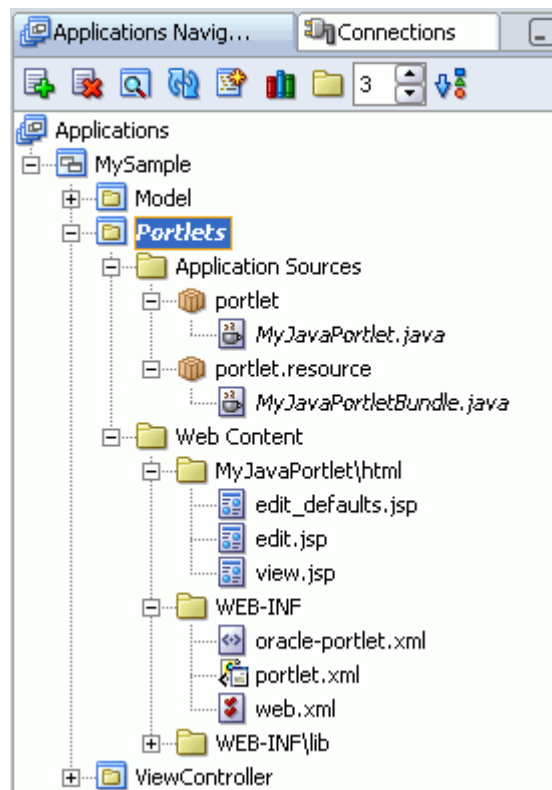
31. On the Portlet Navigation Parameters page, click **Next**.

Navigation parameters are a WSRP 2.0 feature. This page enables you to specify external parameters to be consumed by the JSR 168 portlet. Navigation parameters are not required for this tutorial.

32. Click **Finish**.

After you click **Finish**, you should be able to locate several newly generated files in the Applications Navigator under the Portlets project. The expanded Navigator looks like [Figure 3–13](#).

**Figure 3–13 Files Generated for the New Portlet**



- Under Application Sources, in portlet and portlets.resource, two Java classes:
  - **MyJavaPortlet.java** (under portlet) is invoked by the portlet container. It contains all the methods required by the portlet standards.
  - **MyJavaPortletBundle.java** (under portlet.resource) contains all the translation strings for the portlet.
- Under Web Content, MyJavaPortlet\html:
  - **edit\_defaults.jsp**, which contains the information needed to populate the Customize dialog.
  - **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, two 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 2 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.

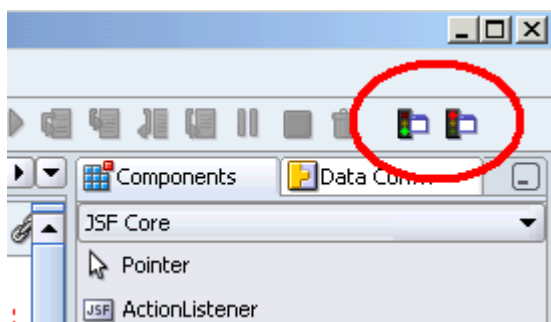
Now that you've created the portlet, it's time to run it to see what it does. To do that, you first must deploy the portlet to an Oracle Application Server. When you installed JDeveloper, you automatically downloaded a standalone OC4J called WebCenter Preconfigured OC4J, so you can use that. However, you first have to establish a connection between this OC4J and the instance of JDeveloper you're using. Let's do that now.

## Step 2: Setting Up Your Connection

Part of deploying a portlet includes establishing a connection to whichever application server you are using. In this tutorial, your portlet will be deployed to the WebCenter Preconfigured OC4J. Let's set up a connection to this OC4J.

1. Start the WebCenter Preconfigured OC4J by clicking the **Start WebCenter Preconfigured OC4J** icon at the far right of the JDeveloper toolbar (Figure 3–14).

**Figure 3–14 Starting the Preconfigured OC4J**

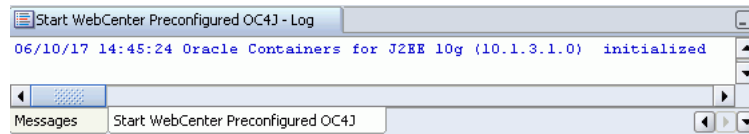


Assuming that this is the first time you have started this OC4J, you will see a message asking if you want to install the WebCenter Preconfigured OC4J.

2. Click **Yes**.

Wait for the message stating that the OC4J instance has initialized before you proceed (see [Figure 3–15](#)).

**Figure 3–15 OC4J Initialization Message**

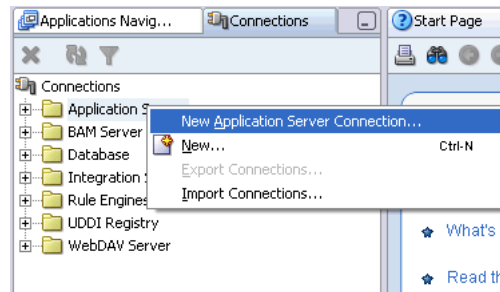


When you first start the preconfigured OC4J, its readme file displays in the JDeveloper Editor. This document describes how to start, stop, test, and connect to the preconfigured OC4J, lists the port defaults, and provides some useful troubleshooting tips. If you need to access this file in the future, select **Help, WebCenter Preconfigured OC4J Readme** from the main JDeveloper menu.

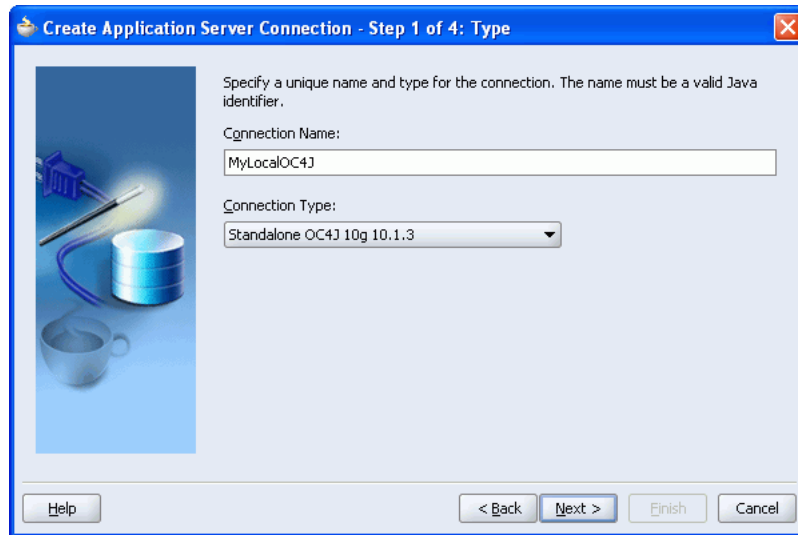
The preconfigured OC4J has been installed on your computer, but you still have to establish a connection to it (so JDeveloper knows where to find it).

3. Click **Connections Navigator** (next to the Applications Navigator).
4. Right-click the **Connections** node, and choose **New Application Server Connection** as shown in [Figure 3–16](#).

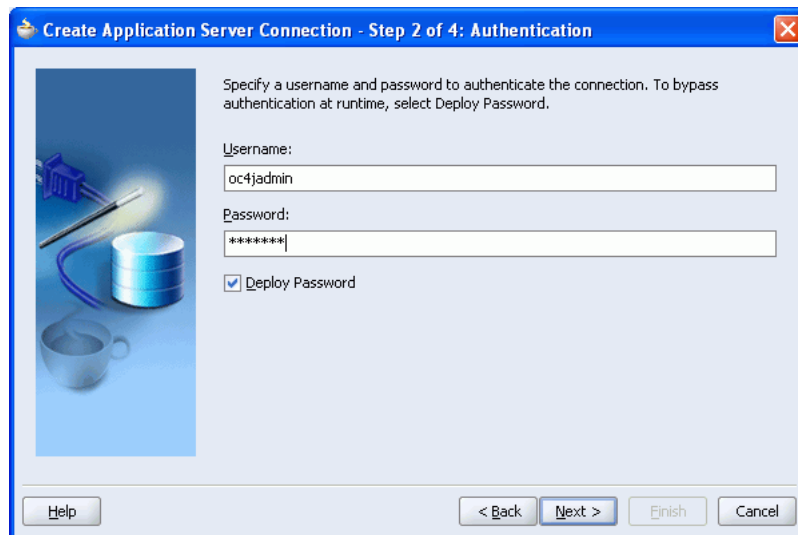
**Figure 3–16 Connecting JDeveloper to the Preconfigured OC4J**



5. Click **Next** to exit the Welcome page.
6. In the Connection Name field, enter: `MyLocalOC4J`
7. From the Connection Type list, select **Standalone OC4J 10g 10.1.3** as shown in [Figure 3–17](#).

**Figure 3–17** *Selecting a Connection Name and Type*

8. Click **Next**.
9. Enter `oc4jadmin` as the username, and the password `welcome`. `welcome` is the default password for the WebCenter Preconfigured OC4J.
10. Select **Deploy Password** to forgo the requirement of a password each time the connection is established (Figure 3–18).

**Figure 3–18** *Entering Login Credentials*

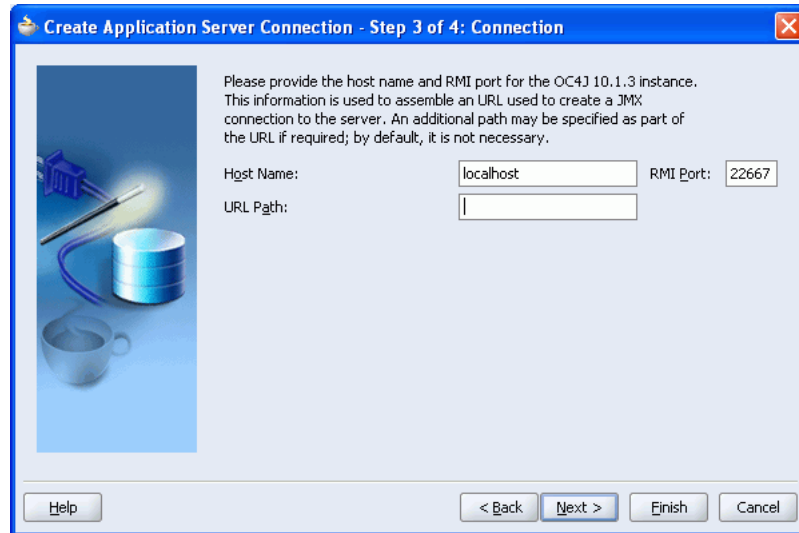
11. Click **Next**.
12. If the WebCenter Preconfigured OC4J resides on your computer, accept the default Host Name `localhost`.

This tutorial has been written on the assumption that `localhost` can be used in a URL to successfully refer to the local computer. However, some firewall configurations can block access to this address. If this occurs, either change your firewall configuration to allow `localhost` or alternatively, enter the fixed IP address of your computer here. If your computer does not have a fixed IP address,

as it is configured to use DHCP, then reconfigure your firewall to allow `localhost` to work.

13. Change the RMI Port to `22667`, the default RMI port for the preconfigured OC4J. You can leave the URL Path blank. This information is not required (Figure 3–19).

**Figure 3–19** Entering Connection Details For the Preconfigured OC4J



14. Click **Next**.
15. Verify the connection by clicking **Test Connection**.

A success message appears in the Status pane if everything is correct. If the test fails, error messages display in the Status field. Use the Back button to return to earlier wizard pages to correct entries.

16. Click **Finish**.

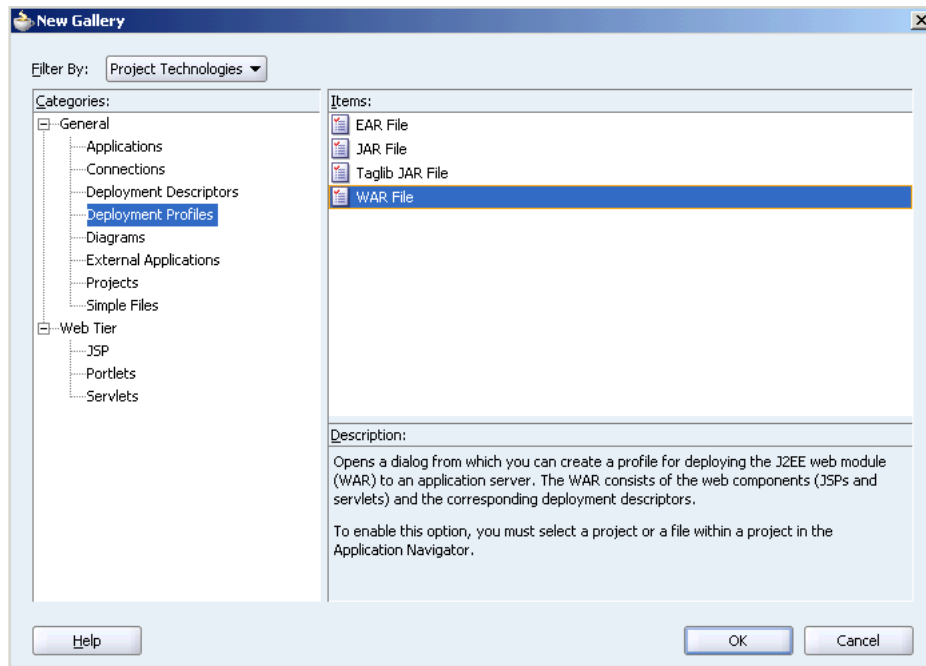
Now that you have established a successful connection, you are ready to include it in a WAR file. WAR stands for *web application archive*, and it packages together all the resources, portlets, and deployment descriptors required to deploy your portlet.

## Step 3: Deploying Your Portlet

In this lesson, you'll learn how to deploy the Java portlet to your local WebCenter Preconfigured OC4J. When you deploy a portlet, you package it up so that it can run on a J2EE server. If you're familiar with OracleAS Portal, we're in effect creating a *portlet provider*, which in the WSRP world is known as a *portlet producer*.

1. Display the **Applications Navigator**.
2. Right-click the **Portlets** project, and click **New**.
3. In the Categories section, expand **General**, and click **Deployment Profiles**.
4. In the Items section, select **WAR File** (Figure 3–20).

**Figure 3–20 Creating a WAR File**

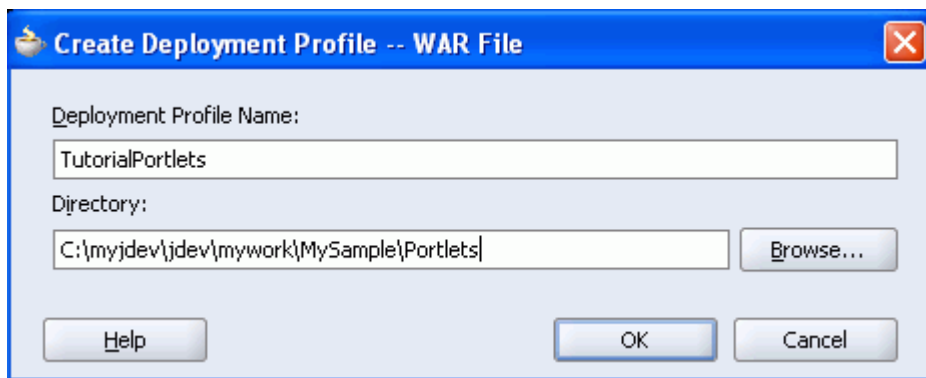


5. Click **OK**.
6. In the Create Deployment Profile - WAR File dialog box (Figure 3–21), enter the following:

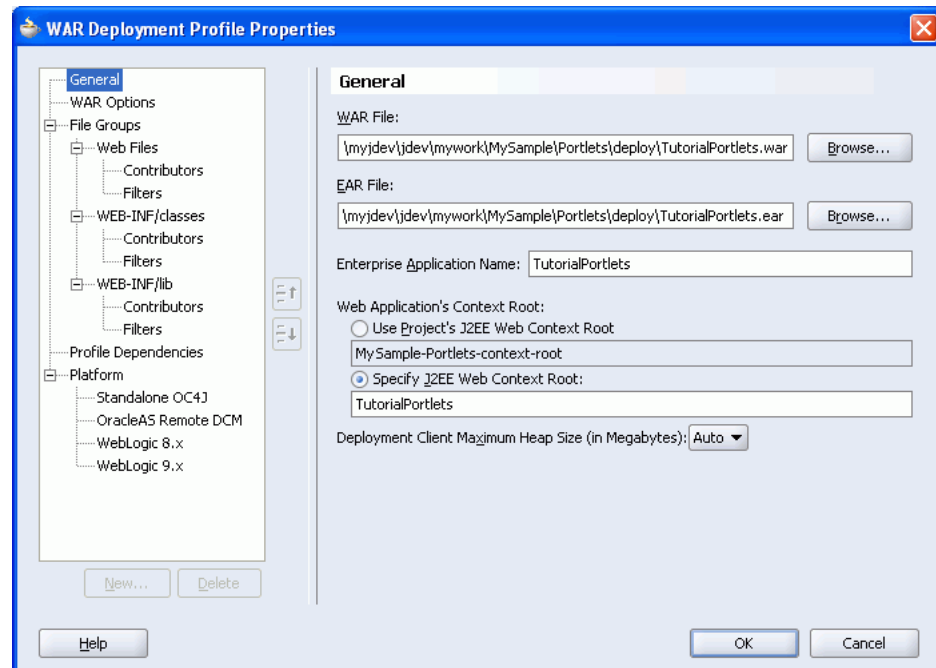
**Table 3–2 Create Deployment Profile - WAR File**

Setting	Value
Deployment Profile Name	Enter: TutorialPortlets
Directory	Navigate to the directory you chose in step 7 in <a href="#">Step 1: Using the JSR 168 Java Portlet Wizard</a> . You should be able to simply accept the default.

**Figure 3–21 Creating a Deployment Profile -WAR File**



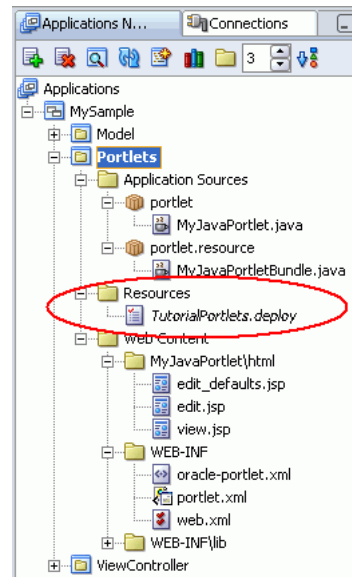
7. Click **OK**.
8. In the WAR Deployment Profile Properties window, select **Specify J2EE Web Context Root**, and then enter TutorialPortlets as shown in Figure 3–22.

**Figure 3–22** Setting WAR Deployment Profile Properties

9. Click **OK**.

10. In the Applications Navigator, expand the **Resources** node.

You should see the deployment profile `TutorialPortlets.deploy` as shown in [Figure 3–23](#).

**Figure 3–23** `TutorialPortlets.deploy` File

11. Right-click **TutorialPortlets.deploy**, and click **Deploy To, MyLocalOC4J**, the connection you created earlier (see [Step 2: Setting Up Your Connection](#)).

12. If the Configure Applications window displays during deployment, click **OK**.

Wait for the *Deployment Finished* message in the Deployment Log (bottom of the JDeveloper window) to verify that the deployment succeeded.

Now let's run the portlet to make sure it's functioning properly.

13. Bring up a browser window, and enter this URL:

`http://<host>:<port>/<context-root>/portlets/wsrp2?WSDL`

where:

Parameter	Value
host	The host name you used to connect to the WebCenter Preconfigured OC4J. Earlier in this lesson, we accepted the default, localhost.  (If you chose to use your computer's IP address, rather than localhost, enter this here instead.)
port	The HTTP Listener port. For the WebCenter Preconfigured OC4J, use 6688, the default port on which this preconfigured OC4J listens. <sup>1</sup>
context-root	TutorialPortlets, as established in step 8 in this lesson.

<sup>1</sup> To change the port on which the preconfigured OC4J listens, go to `JDEVHOME\jdev\extensions\oracle.adf.see.doc4j.10.1.3.2.0\jee\home\config\default-web-site.xml` and change the following entry: `<web-site .... port="6688" ...>`

So, for this tutorial the URL looks something like this:

`http://localhost:6688/TutorialPortlets/portlets/wsrp2?WSDL`

Make a note of this URL. You'll need it in [Step 5: Registering Your Portlet with Oracle WebCenter Framework](#).

14. In the browser, you should now see XML similar to that shown in [Figure 3–24](#).

**Figure 3–24 WSDL Describing Your Portlet as a Web Service**

```

- <wsdl:definitions targetNamespace="urn:oasis:names:tc:wsrp:v2:wsdl">
  <import namespace="urn:oasis:names:tc:wsrp:v2:bind" location="wsrp_v2_bindings.wsdl"/>
  - <wsdl:service name="WSRP_v2_Service">
    - <wsdl:port binding="bind:WSRP_v2_ServiceDescription_Binding_SOAP" name="WSRP_v2_ServiceDescription_Service">
      <soap:address location="http://localhost:6688/TutorialPortlets/portlets/WSRP_v2_ServiceDescription_Service"/>
    </wsdl:port>
    - <wsdl:port binding="bind:WSRP_v2_Markup_Binding_SOAP" name="WSRP_v2_Markup_Service">
      <soap:address location="http://localhost:6688/TutorialPortlets/portlets/WSRP_v2_Markup_Service"/>
    </wsdl:port>
    - <wsdl:port binding="bind:WSRP_v2_Registration_Binding_SOAP" name="WSRP_v2_Registration_Service">
      <soap:address location="http://localhost:6688/TutorialPortlets/portlets/WSRP_v2_Registration_Service"/>
    </wsdl:port>
    - <wsdl:port binding="bind:WSRP_v2_PortletManagement_Binding_SOAP" name="WSRP_v2_PortletManagement_Service">
      <soap:address location="http://localhost:6688/TutorialPortlets/portlets/WSRP_v2_PortletManagement_Service"/>
    </wsdl:port>
  </wsdl:service>
</wsdl: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. Assuming your WSDL looks something like [Figure 3–24](#), the next step is to create a JSF page that will consume the portlet.

## Step 4: Creating a JSF Page

To create a JavaServer Faces (JSF) page:

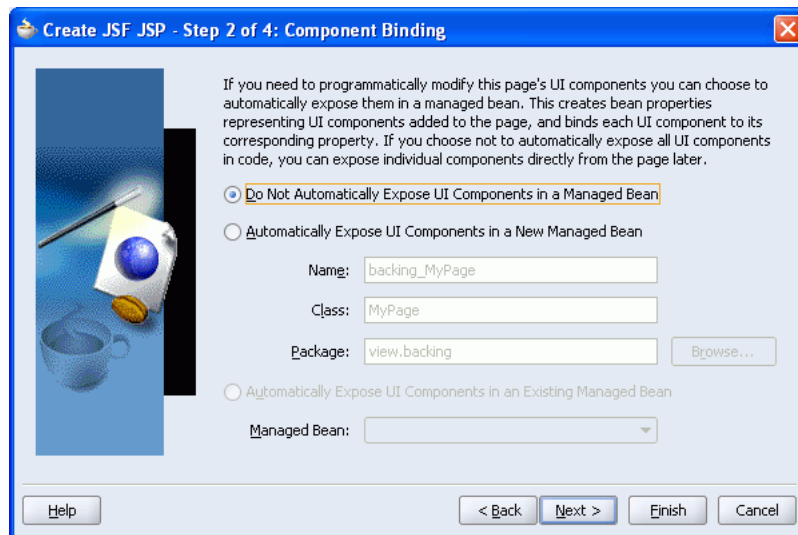


1. In the Applications Navigator, right-click the **ViewController** project, and select **New**.
2. In the Categories pane, under Web Tier, click **JSF** to create a JavaServer Faces page.
3. Under Items, select **JSF JSP**, and click **OK**.  
The JSF JSP Wizard opens.
4. Click **Next** to exit the Welcome page.
5. In the File Name field as shown in [Figure 3–25](#), enter: `MyPage`  
You can accept the default location for the Directory Name.
6. Under Type, click **JSP Document** so an XML version of the page (that is, a .jspx file) will be created.

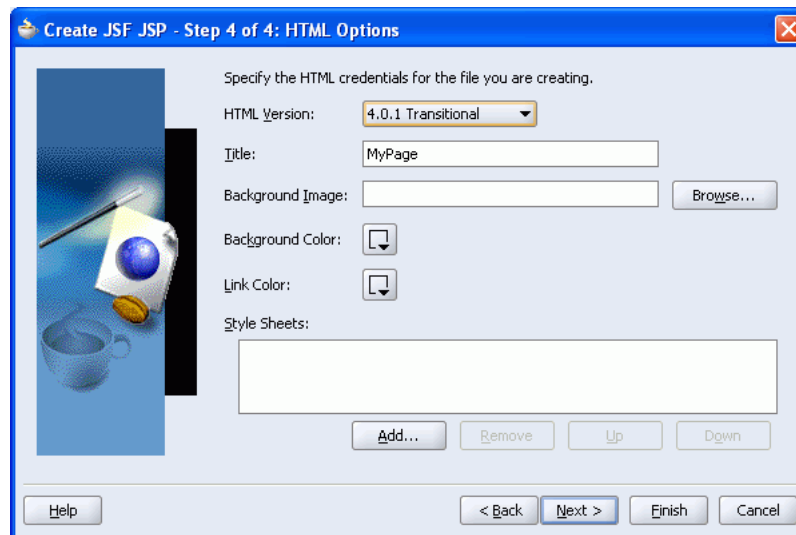
**Figure 3–25** *Creating a JSP Page*



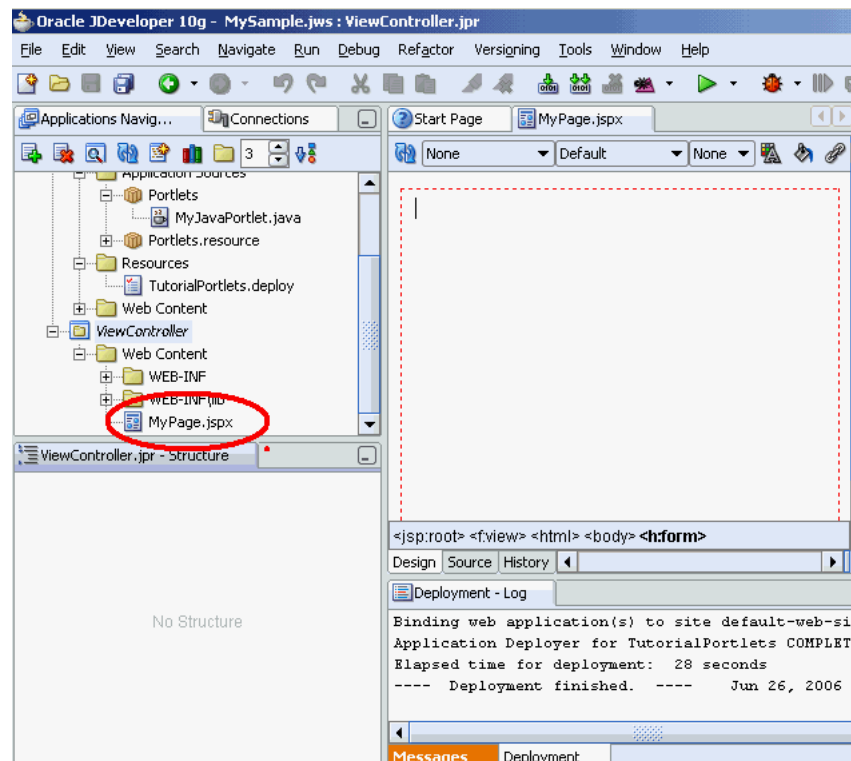
7. Click **Next**.
8. We won't be adding any back-end logic to this page (at least, not in this lesson), so we don't need a new managed bean. Leave the default setting as is (**Do Not Automatically Expose UI Components in a Managed Bean**) as shown in [Figure 3–26](#).

**Figure 3–26 Component Binding Page of the Create JSF JSP Wizard**

9. Click **Next**.
10. On the Tag Libraries page, use the double arrows to move all the libraries from the Available Libraries pane to the Selected Libraries pane. The following libraries should be included:
  - ADF Faces Components 10\_1\_3\_2\_0
  - ADF Faces HTML 10\_1\_3\_2\_0
  - ADF Portlet Components 10\_1\_3\_2\_0
  - Customizable Components Core 10\_1\_3\_2
  - JSF Core 1.0
  - JSF HTML 1.0
11. Click **Next**.
12. We don't need any special HTML options, so click **Finish** on the page shown in [Figure 3–27](#).

**Figure 3–27 Creating a JSF JSP - HTML Options**

You should now be able to see the page as shown in [Figure 3–28](#) you just created in the Applications Navigator, under ViewController, Web Content, MyPage.jspx.

**Figure 3–28 MyPage in the Applications Navigator**

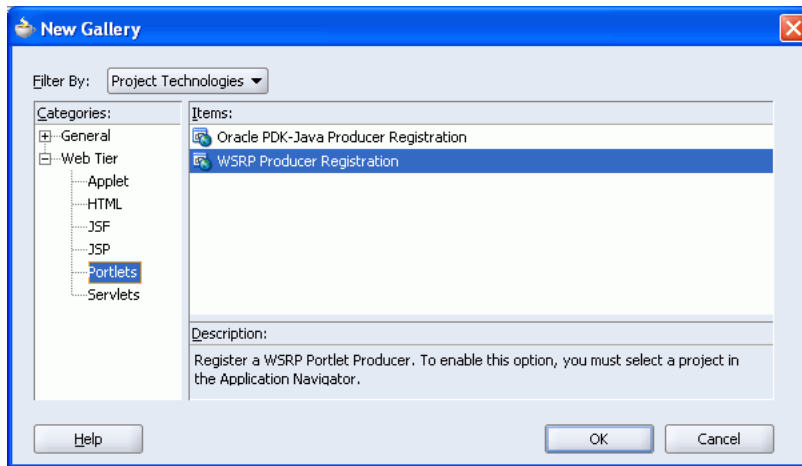
You should also see your page open in the Visual Editor, ready for us to begin adding components.

## Step 5: Registering Your Portlet with Oracle WebCenter Framework

So far we have created a portlet, a portlet producer, and a JavaServer Faces page. Now we need to tie them all together and tell the page how to access the producer. This is also known as registering the producer with the application.

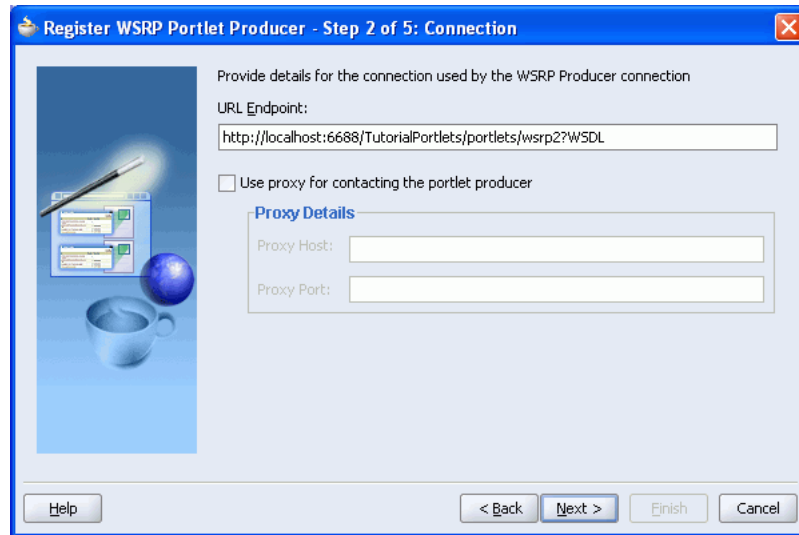
1. Right-click **ViewController** in the Applications Navigator, and click **New**.
2. In the New Gallery, under Web Tier, click **Portlets** as shown in [Figure 3–29](#).

**Figure 3–29** New WSRP Producer Registration



3. Select **WSRP Producer Registration**, and click **OK**.  
This opens the WSRP Portlet Producer Registration Wizard.
4. Click **Next** to exit the Welcome page.
5. In the Name field, enter: `TutorialProducer`
6. Click **Next**.
7. On the Connection page ([Figure 3–30](#)), enter the URL you constructed in [Step 3: Deploying Your Portlet](#) (sub step 13). For example:

```
http://localhost:6688/TutorialPortlets/portlets/wsrp2?WSDL
```

**Figure 3–30 New WSRP Producer Connection Details**

Proxy information isn't necessary for this tutorial as you're using a local preconfigured OC4J.

8. Click **Next**.
9. Click **Next** again to accept the default timeout value of 30 seconds.
10. Click **Next** to accept the default security settings.
11. On the Key Store page, click **Finish**.

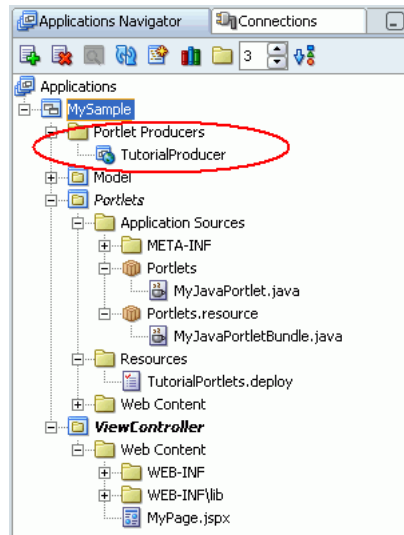
You should see a message that looks like [Figure 3–31](#).

**Figure 3–31 Successfully Registered Portlet Producer**

12. Click **OK** to close this message box.

A Portlet Producers node now displays in the Applications Navigator as shown in [Figure 3–32](#). Expand this node and verify that TutorialProducer appears there.

**Figure 3–32** *TutorialProducer in the Applications Navigator*



The page now knows where to locate the portlet `MyJavaPortlet` and how to access it. Let's verify to make sure.

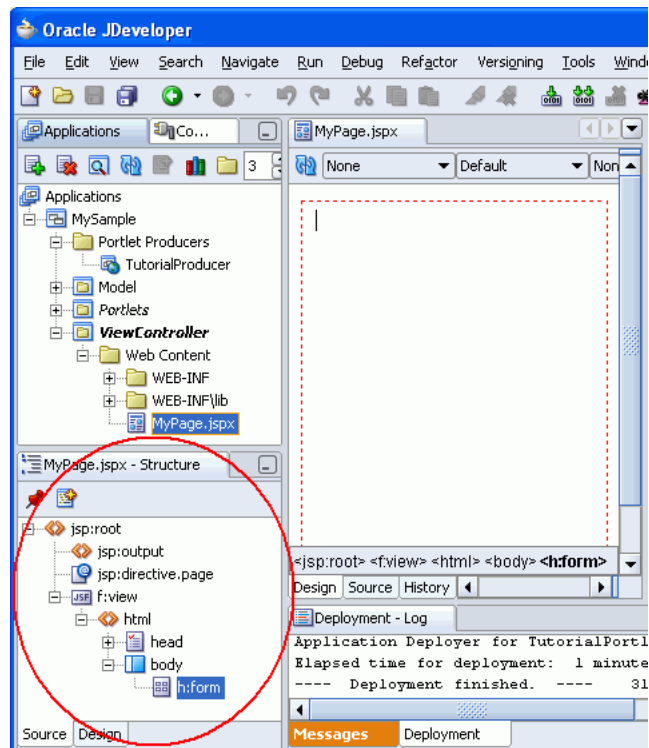
## Step 6: Testing the Portlet

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

1. If `MyPage.jspx` isn't already open, locate the page name in the Applications Navigator (under `ViewController`, `Web Content`) and double-click it.

This opens the page in Structure window, as shown in [Figure 3–33](#).

Figure 3–33 MyPage.jspx Open in Structure Window

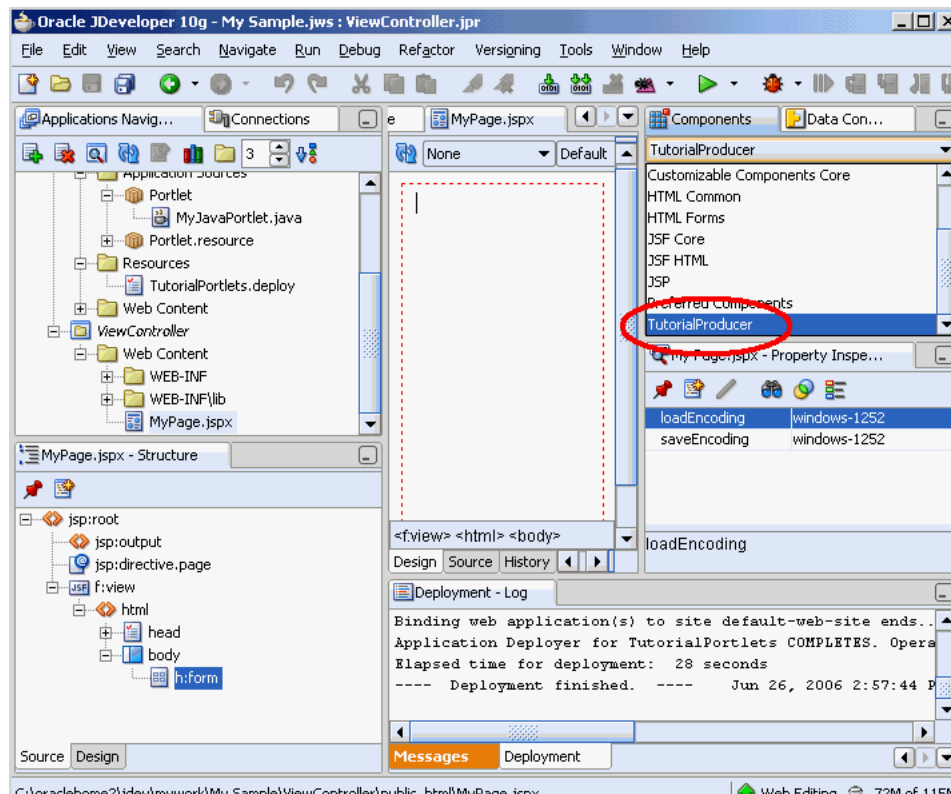


2. On the far right side of the JDeveloper window, click the Component Palette drop-down list as shown in Figure 3–34, and verify that TutorialProducer is listed there.

---

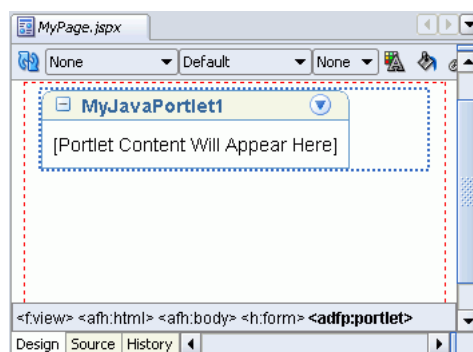
**Note:** Sometimes the Component Palette becomes hidden. If you don't see it, select View, Component Palette from the JDeveloper menu.

---

**Figure 3–34 TutorialProducer in the Component Palette**

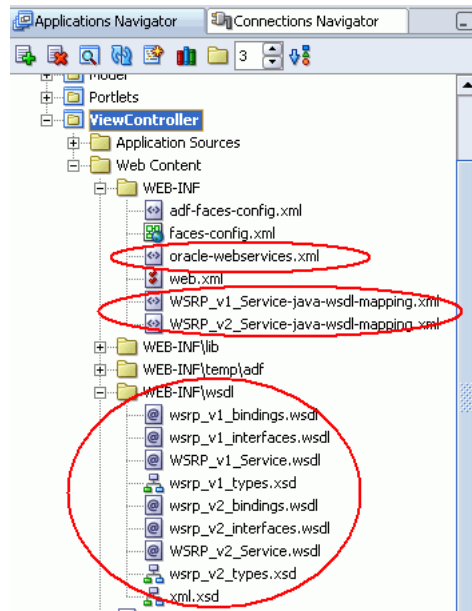
3. Select **TutorialProducer**. You should see your portlet `MyJavaPortlet` listed there.
4. Select **MyJavaPortlet**, and drag it across to `h:form`, the last entry in the Structure window. You want to make sure that the portlet is included IN the form, and the Structure window makes it easy to do this.

You should now see something that looks like [Figure 3–35](#).

**Figure 3–35 MyJavaPortlet Added to MyPage.jspx**

You should be able to locate several newly generated files and folders in the Applications Navigator. The expanded Navigator looks like [Figure 3–36](#). These internal files are created when consuming portlets of WSRP producers. You won't need to edit any of these files.



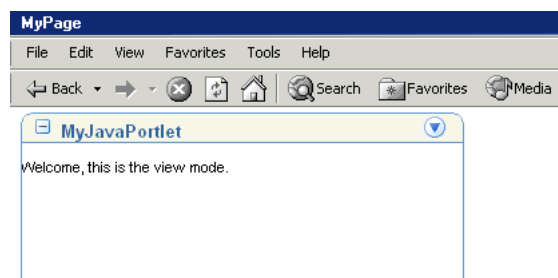
**Figure 3–36** WEB-INF\wsdl Created in Application Navigator

Now let's run the page.

5. In the Applications Navigator, right-click **MyPage.jspx**, and select **Run**.

This may take a few moments. You may notice that the Message Log says Running: Embedded OC4J Server. While the portlet is running in the preconfigured OC4J you downloaded at the beginning of this tutorial, the page itself is running in JDeveloper's embedded OC4J.

The page should open in a new browser window ([Figure 3–37](#)).

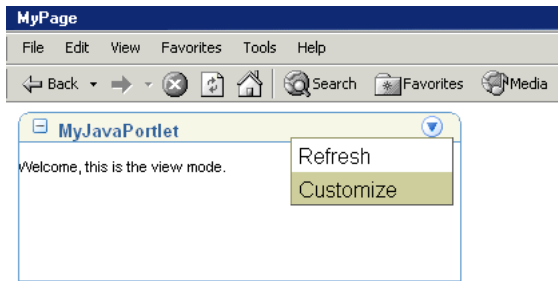
**Figure 3–37** MyJavaPortlet in a Browser Window

6. Click the **Actions** menu on the portlet header.

You should see two options: Refresh and Customize ([Figure 3–38](#)). Notice that the Personalize option does not display. MyJavaPortlet *does* support user personalizations but you'll only see the Personalize option when the portlet appears in an application that implements a security model and you're logged in with valid user credentials. When we get to [Chapter 8, "Providing Security"](#), we'll test the Personalize mode.

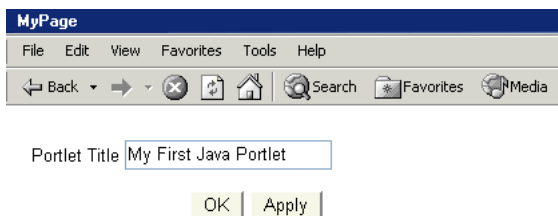
For now, let's test the Customize mode.

**Figure 3–38 Actions for MyJavaPortlet**



7. From the Actions drop-down menu, select **Customize**.
8. In the Portlet Title field, change the portlet header to something else, such as My First Java Portlet as shown in [Figure 3–39](#).

**Figure 3–39 Changing the Portlet Title**



9. Click **OK**.  
 The portlet is redisplayed in the browser, with the name you chose displayed in the header.  
 We now know that this portlet is working properly, but it isn't very interesting. Let's now add some logic so that users can paste HTML into a text box for the portlet to render.

## Step 7: Adding Some Simple Logic to the Portlet

In this step, we'll add some code that allows a user to paste HTML into a text box for the portlet to render.

1. Return to JDeveloper.
2. In the Applications Navigator, under Portlets, Web Content, MyJavaPortlet\html, double-click **view.jsp**.  
 A new editing window opens to the right of the Applications Navigator.
3. Click the **Source** tab at the bottom of the new window.
4. To the existing JSP code, add the lines highlighted in [Figure 3–40](#):

**Figure 3–40 Edits to view.jsp**

```

<%@ page contentType="text/html"
    pageEncoding="windows-1252"
    import="javax.portlet.*,java.util.*,portlet.MyJavaPortlet,portlet.resource.MyJavaPortletBundle"%>
<%@ taglib uri="http://java.sun.com/portlet" prefix="portlet"%>
<portlet:defineObjects/>
<%String[] str = {"Portlet Content"};
PortletPreferences prefs = renderRequest.getPreferences();
str = prefs.getValues("portletContent",str);
for (int i=0; i<str.length; i++)
{
%><%= (i<str.length-1)?str[i]+", ":str[i]%><%=
<p class="portlet-font">Welcome, this is the <%= renderRequest.getPortletMode().toString() %> mode.</p>

```

Here is the code to copy and paste:

```

<%
String[] str = {"Portlet Content"};
PortletPreferences prefs = renderRequest.getPreferences();
str = prefs.getValues("portletContent",str);
for (int i=0; i<str.length; i++)
{
%><%= (i<str.length-1)?str[i]+", ":str[i]%><%=

```

This code will retrieve the preferences set in Personalize or Customize mode and display them in the portlet's View mode so that users can see them.

5. Click the **Save** icon in JDeveloper's toolbar.
6. Double-click **edit.jsp** to open it in the Visual Editor.
7. Click the **Source** tab.
8. Add the code as shown in [Figure 3–41](#) to implement a form field called Content. Here is the code to copy and paste:

```

<%
    String[] str = {"Portlet Content"};
    str = prefs.getValues("portletContent",str);
%>
<tr><td width="20%">
    <p class="portlet-form-field" align="right"> Content</p>
</td><td width="80%">
    <textarea rows="10" cols="60" class="portlet-form-input-field"
name="portletContent"><%
    for (int i=0; i<str.length; i++)
    {%><%= (i<str.length-1) ? str[i]+", " : str[i] %><%=
</textarea>
</td></tr>

```

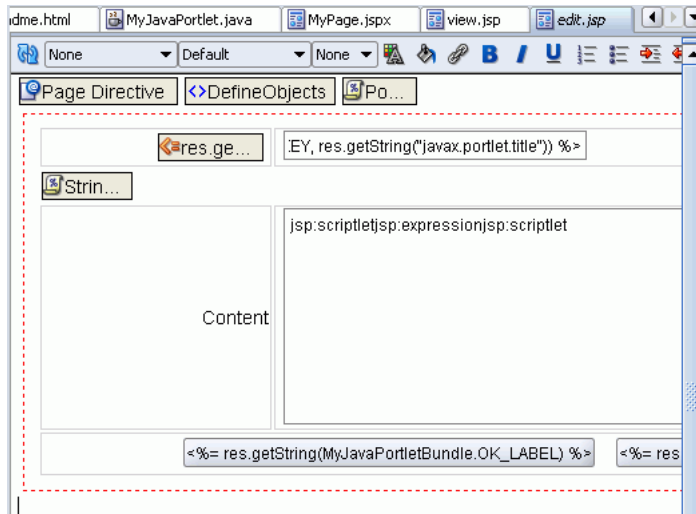
**Figure 3–41 Adding Code to edit.jsp**

```

<FORM ACTION="<portlet:actionURL/>" METHOD="POST">
<TABLE BORDER="0">
<TR><TD WIDTH="20%">
<P CLASS="portlet-form-field" ALIGN="right">
<%= res.getString(MyJavaPortletBundle.PORTLETTITLE) %>
</P></TD><TD WIDTH="80%">
<INPUT CLASS="portlet-form-input-field" TYPE="TEXT"
NAME="<%= MyJavaPortlet.PORTLETTITLE_KEY %>"
VALUE="<%= prefs.getValue(MyJavaPortlet.PORTLETTITLE_KEY, res.getString("javax.portlet.title")) %>"
SIZE="20">
</TD></TR>
<% String[] str = {"Portlet Content"};
   str = prefs.getValues("portletContent",str); %>
<tr><td width="20%">
<p class="portlet-form-field" align="right"> Content</p>
</td><td width="80%">
<textarea rows="10" cols="60" class="portlet-form-input-field" name="portletContent"><%
   for (int i=0; i<str.length; i++)
   {><%= (i<str.length-1) ? str[i]+", " : str[i] %><%>
</textarea>
</td></tr>
<TR><TD COLSPAN="2" ALIGN="CENTER">
<INPUT CLASS="portlet-form-button" TYPE="SUBMIT" NAME="<%= MyJavaPortlet.OK_ACTION %>"
VALUE="<%= res.getString(MyJavaPortletBundle.OK_LABEL) %>">
<INPUT CLASS="portlet-form-button" TYPE="SUBMIT" NAME="<%= MyJavaPortlet.APPLY_ACTION %>"
VALUE="<%= res.getString(MyJavaPortletBundle.APPLY_LABEL) %>">
</TD></TR>
</TABLE>
</FORM>

```

9. Click the **Save** icon in JDeveloper's toolbar.
10. Click the **Design** tab to see the form field you just added as shown in [Figure 3–42](#).

**Figure 3–42 Adding the Content Field**

You've edited the file that will store the changes made through Personalize mode, but now you must make the same changes to the file that stores the changes made through Customize mode—that is, `edit_defaults.jsp`.

11. Double-click `edit_defaults.jsp`, and add the same code in the same location. Don't forget to click the **Save** icon when you are through.

`MyJavaPortlet.java`, the file containing the java code for this portlet, must now be edited as well.

12. In the Applications Navigator, under Portlets, Application Sources, portlet, double click **MyJavaPortlet.java** to open it in the Visual Editor's Source view.
13. Scroll down to the `processAction` method and locate the code line that says `//Save the preferences`. Insert the following two lines of code (indicated in bold):

```
// Save the preferences.
PortletPreferences prefs = request.getPreferences();
String param = request.getParameter(PORTLETTITLE_KEY);
prefs.setValues(PORTLETTITLE_KEY, buildValueArray(param));
String contentParam = request.getParameter("portletContent");
prefs.setValues("portletContent", buildValueArray(contentParam));
prefs.store();
```

14. Click the **Save All** icon to save your changes to `view.jsp`, `edit.jsp`, `edit_defaults.jsp`, and `MyJavaPortlet.java`.

Because you've made some changes, you need to redeploy the portlet to your preconfigured OC4J.

15. In the Applications Navigator, under Portlets, Resources, right-click **TutorialPortlets.deploy**.

16. Click **Deploy to MyLocalOC4J**.

17. Click **OK** to dismiss the Configure Application dialog.

18. Because `TutorialPortlets.deploy` was already deployed, click **Yes** to confirm undeploying the original version.

Wait for the `Deployment Finished` message. Notice how JDeveloper automatically saves and compiles the code before deploying the portlet.

19. Before running the page again, stop the embedded OC4J Server. Choose **Run, Terminate, Embedded OC4J Server** from the main menu.

Alternatively, click the red square Terminate icon in the Embedded OC4J Server Log window. It is good practice to stop the embedded OC4J server before testing your pages. JDeveloper will automatically restart the embedded OC4J server, when you run a page.

20. Make sure that `MyPage.jspx` appears in the Visual Editor, then click **Run > MyPage.jspx**.

A new browser window opens displaying `MyJavaPortlet`.

21. Select **Customize** from the drop-down menu.

Because we added the Content field, the Customize action now displays a text box that looks like [Figure 3-43](#).

**Figure 3–43 New Customize Option**

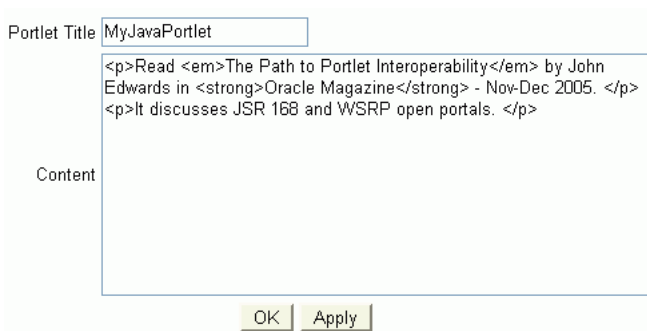


22. Change the Portlet Title back to `MyJavaPortlet`.

23. Copy and paste the following HTML text in the Content field as shown in [Figure 3–44](#).

```
<p>Read <em>The Path to Portlet Interoperability</em> by John Edwards in
<strong>Oracle Magazine</strong> - Nov-Dec 2005. </p>
<p>It discusses JSR 168 and WSRP open portals. </p>
```

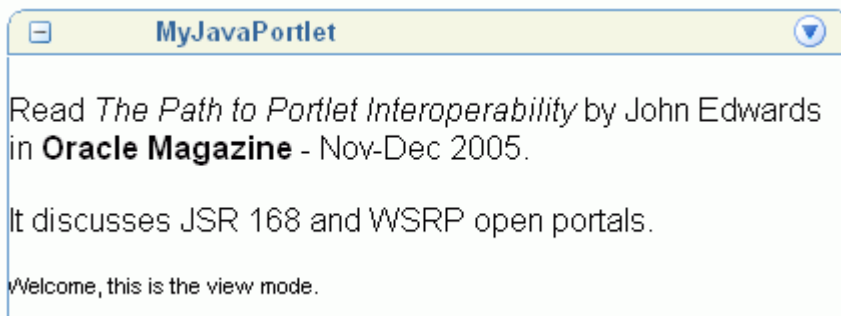
**Figure 3–44 Customize Portlet Title and Content**



24. Click **OK**.

Your browser should now look like [Figure 3–45](#).

**Figure 3–45 New HTML Text in MyJavaPortlet**



If you want to remove the default text (*Welcome, this is the view mode.*), simply double-click **view.jsp**, remove the final line of code, redeploy the portlet to your preconfigured OC4J, then re-run the page.

In the next lesson, you'll add some additional content to your page in the form of images.





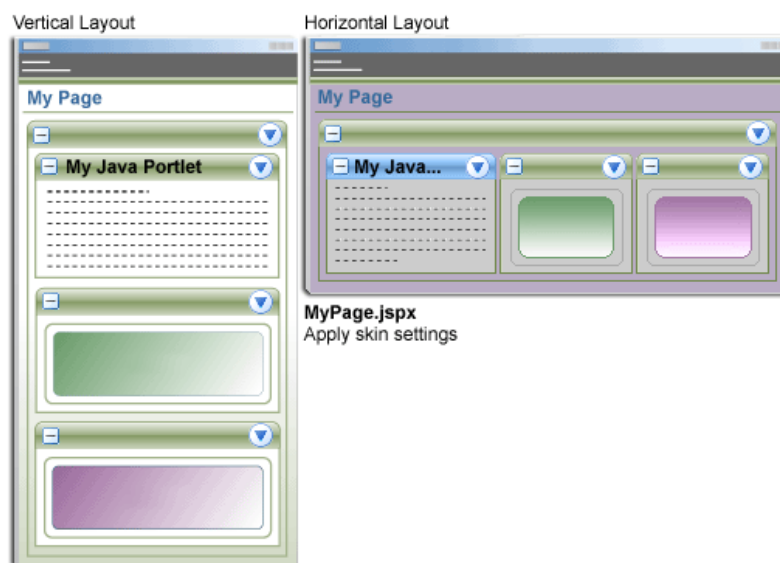
---

## Customizing Your Page

In this lesson you will extend the page created in the previous lesson by adding customizable components to the page and experimenting with ways to affect the component's look and feel. Along the way you will test your page to confirm that your page is developing the way you expect it to.

Figure 4–1 shows how the components on your page will be laid out once you complete this lesson.

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



### Introduction

We will customize the page by completing the following steps:

- [Step 1: Allowing Users To Customize the Page](#)
- [Step 2: Running and Customizing the Page](#)
- [Step 3: Making Additional Customizations](#)
- [Step 4: Testing the New Customizations](#)
- [Step 5: Changing the Look and Feel](#)

Let's begin by examining some of the benefits that Oracle WebCenter Framework provides.

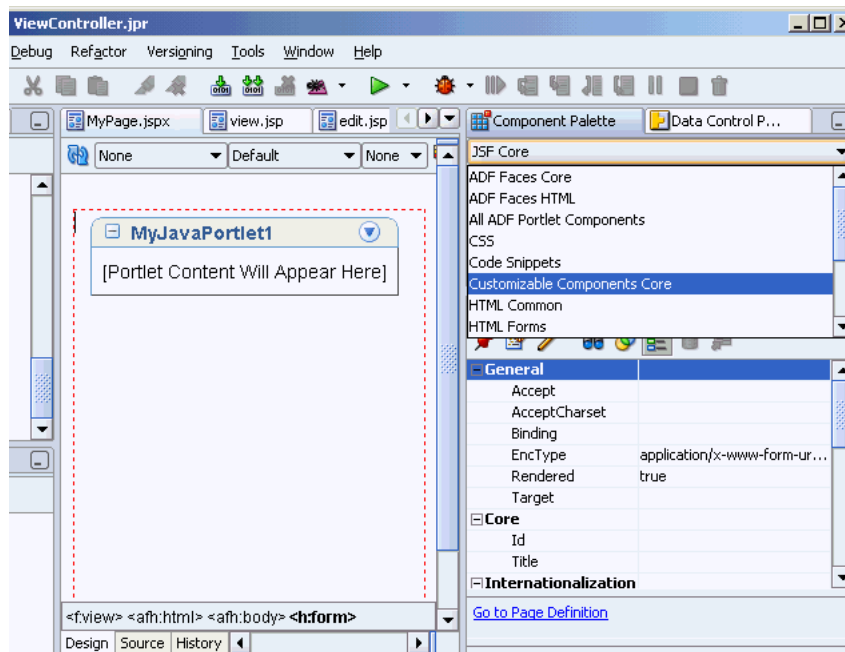
## Step 1: Allowing Users To Customize the Page

One of the key features of Oracle WebCenter Framework is the concept of "Design Time at Run Time", which allows site administrators to customize the page for others. Customization is achieved through *customizable components*, specifically `PanelCustomizable` and `ShowDetailFrame`. (Customizing *pages* is different from customizing *portlets*, which you did using portlet actions in the last chapter.)

Using these two components, you can give users the option of hiding a certain piece of content, or moving it to a different location on the page. Let's see how this works.

1. Make sure `MyPage.jspx` appears in the Visual Editor.
2. Open the Component Palette.
3. Using the drop-down list, select **Customizable Components Core** (Figure 4-2).

**Figure 4-2 Customizable Components Core**



There are two items in this group:

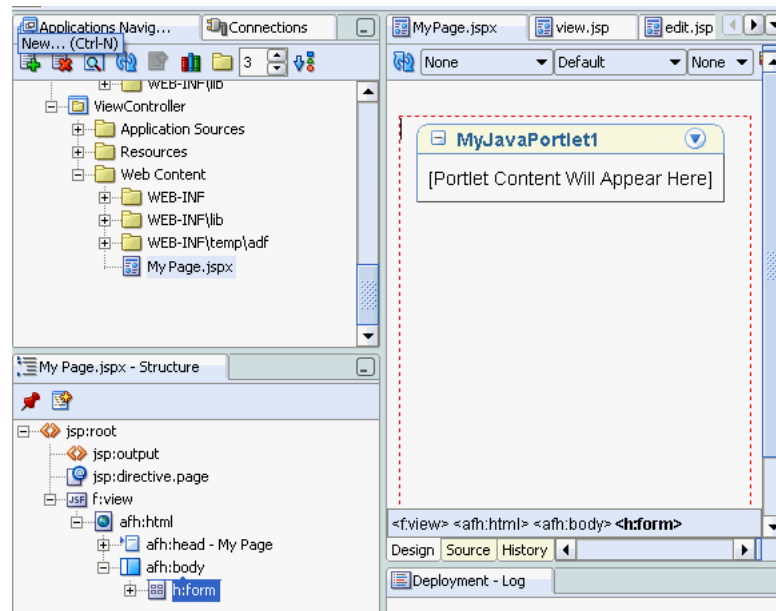
- **PanelCustomizable**, which provides a horizontal or vertical layout in which to include Oracle ADF components. By setting a few attributes in the Property Inspector, you can enable actions for showing or hiding whatever content is placed within the layout.
- **ShowDetailFrame**, through which you can display the actions allowed for a particular component. In this release of Oracle WebCenter Framework, users can move, minimize, or maximize content. You can think of `ShowDetailFrame` as the *chrome*, or border that goes around the components on the page.

Let's see these concepts in action.

4. In the Applications Navigator, highlight `MyPage.jspx`.

Notice how the structure of the page appears in the Structure window (Figure 4-3).

Figure 4-3 Structure of MyPage.jspx



5. In the Component Palette, highlight **PanelCustomizable**, and drag it to the Structure window. Drop it over the `h:form` tag.

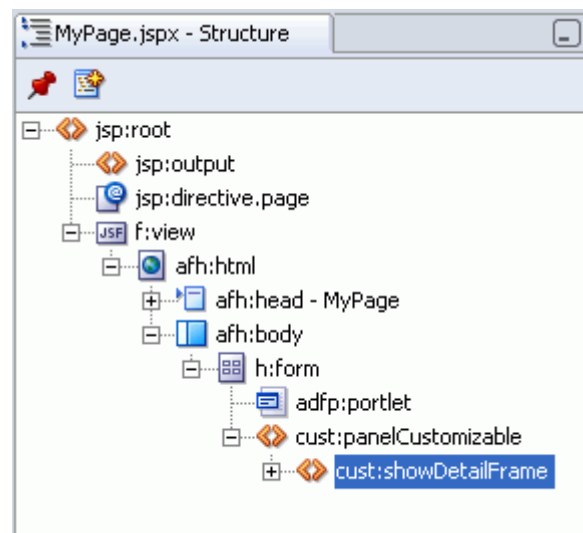
**Tip:** When you drop a component inside a form, make sure that `h:form` has a box around it.

Working with the Structure view provides greater certainty where a component is being placed but you can also drag and drop components directly onto the Visual Editor.

6. In the Component Palette, highlight **ShowDetailFrame**, and drag it to the Structure window. Drop it over `cust:panelCustomizable`.

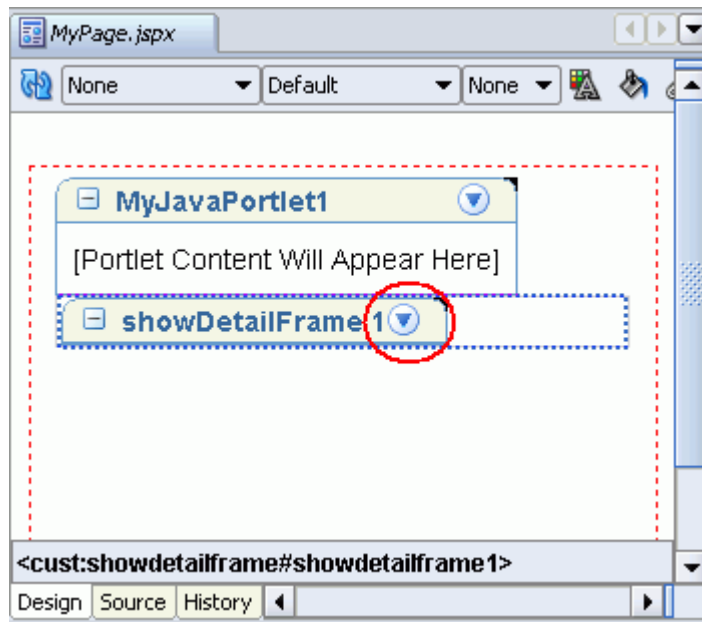
`cust:showDetailFrame` becomes a child of `cust:panelCustomizable`, as shown in Figure 4-4.

Figure 4-4 Hierarchy in Structure Window



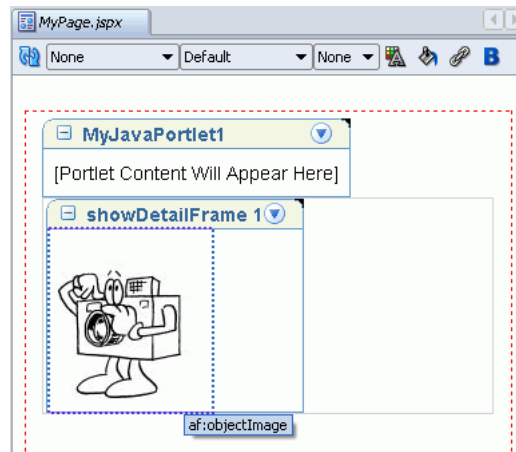
In the Visual Editor (shown in [Figure 4-5](#)), you can see `showDetailFrame1` within a grey box. The grey box indicates the `cust:panelCustomizable`, so we know that we safely dropped `showDetailFrame` within it. The area that is circled represents the control that users will use to customize the content of `showDetailFrame`.

**Figure 4-5** *ShowDetailFrame Component Added to MyPage.jspx*



Let's add some content now, in the form of a simple image.

7. In the Component Palette drop-down list, select **ADF Faces Core**.  
This presents you with a list of commonly used ADF components.
8. Scroll down the alphabetical list until you see the **ObjectImage** component.
9. Highlight **ObjectImage**, and drag it to the Structure window. Drop it on top of `cust:showDetailFrame`.  
The Insert ObjectImage window opens.
10. Use the ... button to locate the directory where you unzipped `webcentertutorialcontent.zip` (see [Downloading Sample Tutorial Files](#)).  
For example: `C:\TutorialContent`
11. Highlight the image `camera.gif`, and click **OK**.  
A dialog opens asking you if you want to locate the image under the document root. If you choose Yes, a copy of the image is saved to a location that is automatically included when you create the application deployment archive.
12. Click **Yes**.
13. Click **Save**, then **OK**.

**Figure 4-6** Adding an Image to MyPage.jspx

The image `camera.gif` appears inside the `showDetailFrame` component as shown in [Figure 4-6](#).

**Tip:** You can drag and drop images directly from your file system, for example `C:\TutorialContent\camera.gif`, onto a `ShowDetailFrame` component. An `ObjectImage` is created automatically, and the image is copied to the project.

Let's add another image so you can see how users will be able to move the images around the page.

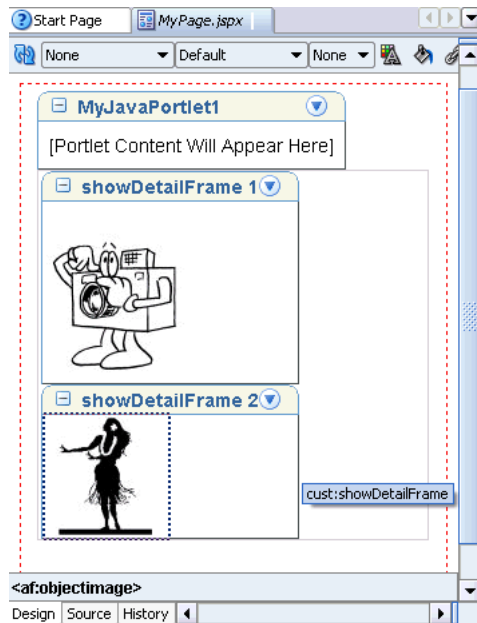
14. In the Component Palette drop-down list, select **Customizable Components Core**, and select **ShowDetailFrame**.
15. Drag **ShowDetailFrame** to the Structure window and drop it on `cust:panelCustomizable`.
16. In the Component Palette drop-down list, select **ADF Faces Core**, and select **ObjectImage**.
17. Drag **ObjectImage** to the Structure window. Drop it on top of the second `cust:showDetailFrame`.
18. Use the ... button to locate `hula.gif` (see [Downloading Sample Tutorial Files](#)), and click **OK**.

A dialog opens asking you if you want to locate the image under the document root.

19. Choose **Yes**.
20. Click **Save**, then **OK**.

The Design tab in the Visual Editor should now look something like [Figure 4-7](#).

Figure 4-7 Two Images and a Portlet on MyPage.jspx



Each image has its own actions icon on the `showDetailFrame` header, so each can be acted upon independently.

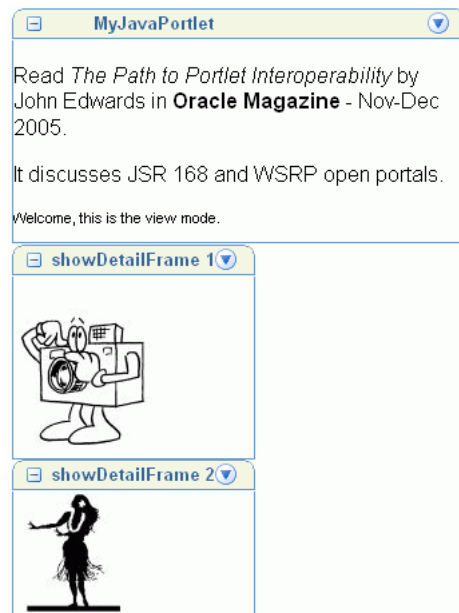
Now let's run the page and see what kind of customizations are possible.

## Step 2: Running and Customizing the Page

In this step, we run the page and test the customizations that a user might make.

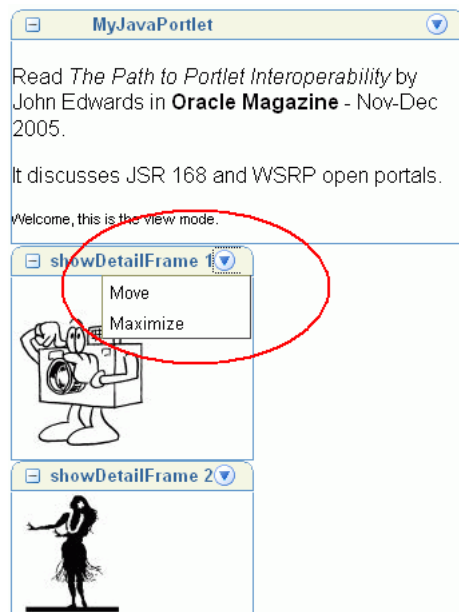
1. Right-click **MyPage.jspx**, and click **Run**.

Assuming there aren't any compiler errors, the page opens in another browser window, as shown in [Figure 4-8](#).

**Figure 4–8 MyPage.jsp with Two Images**

2. Click the **Actions** icon on the first image.

You should see two actions: Move and Maximize as shown in [Figure 4–9](#).

**Figure 4–9 Actions Menu**

These actions are automatically provided by `showDetailFrame`. Any components you drop within a `showDetailFrame` can be moved or maximized within the context of its parent `panelCustomizable`.

3. Click **Maximize** in the first image and see how its `showDetailFrame` expands to fill the entire `panelCustomizable` as shown in [Figure 4–10](#). The second image is hidden.

Notice that MyJavaPortlet is unaffected by this action. That's because the portlet is not part of this `panelCustomizable`.

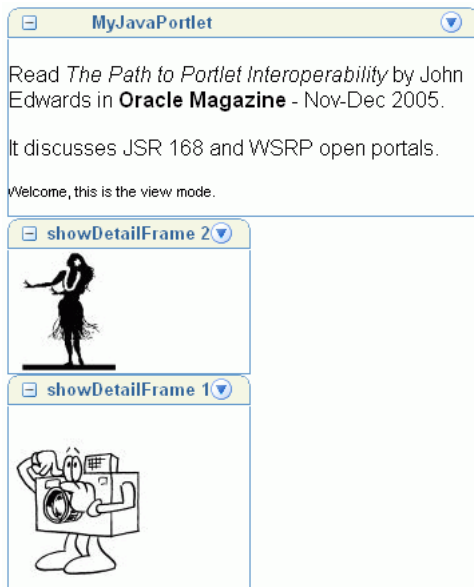
**Figure 4–10 One Image Maximized**



4. Click the **Actions** menu, and then click **Restore** to return the image to its original size.
5. On either of the images, click **Move, Down** or **Move, Up** to see the images switch places on the page. Again, because MyJavaPortlet is not controlled by this `panelCustomizable`, it remains stationary as the images trade places.

Figure 4–11 shows the browser after **Move, Down** was used on the first graphic, the camera. The hula graphic is now on top of the camera graphic.

**Figure 4–11 Moving Images Around**



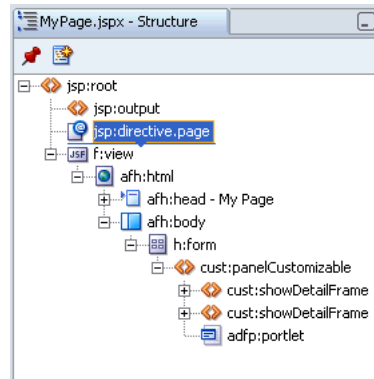
In the next step, we'll move MyJavaPortlet into `cust:panelCustomizable`, and experiment with the Property Inspector to affect the behavior of different components on the page.



## Step 3: Making Additional Customizations

1. Close the browser, and return to JDeveloper.
2. In the Structure window, drag `adfp:portlet`, which represents MyJavaPortlet, and drop it on `cust:panelCustomizable` (Figure 4-12).

**Figure 4-12** *Dropping the Portlet Onto PanelCustomizable*



The portlet automatically goes to the bottom of the list of objects, and the Design view follows suit. The portlet now appears beneath the second image.

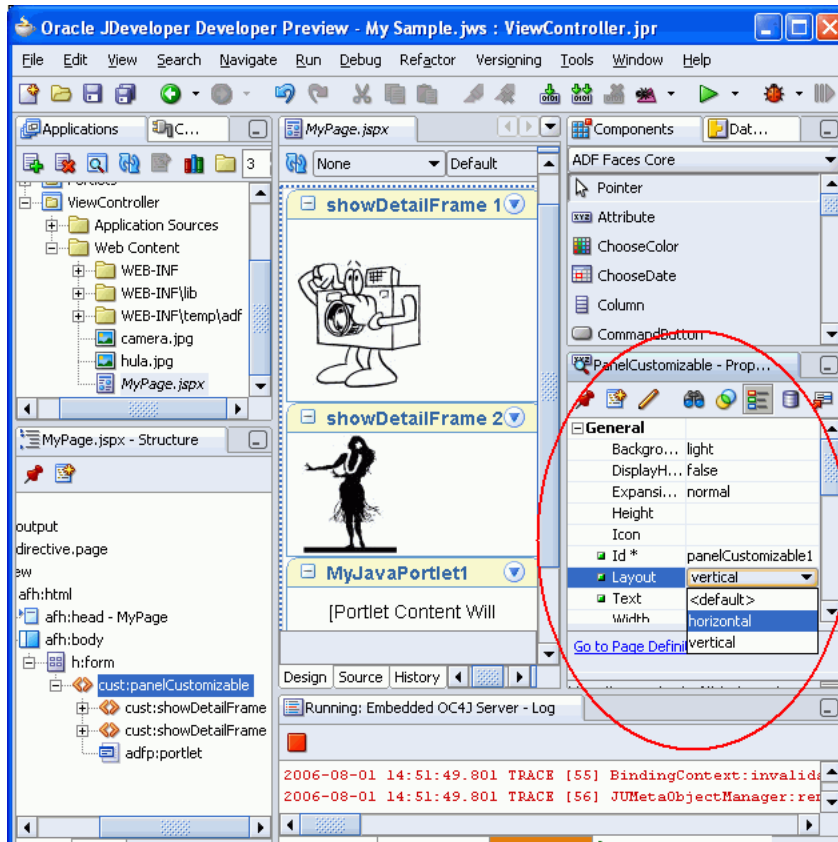
Notice that we dropped the portlet directly onto `cust:panelCustomizable` without inserting a `showDetailFrame` first. That's because the portlet comes automatically equipped with the customization capabilities provided by the portlet chrome; that is, the ability to maximize/minimize and to reposition the portlet on the page.

Now let's choose a different orientation for the portlet and images. Rather than displaying them vertically, let's display them horizontally.

3. In the Structure window, highlight `cust:panelCustomizable`.
4. In the Property Inspector, under General, click the field beside **Layout**.

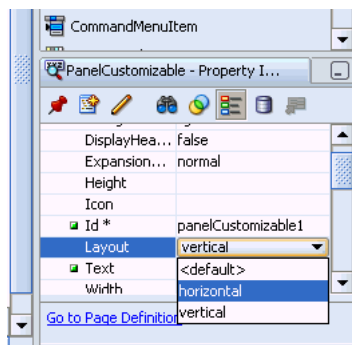
The Property Inspector is to the right of your JDeveloper screen (Figure 4-13).

**Figure 4–13 JDeveloper Property Inspector**



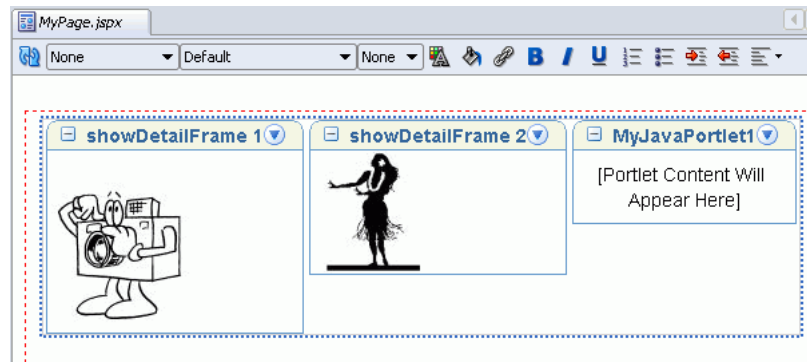
5. Select **horizontal** as shown in [Figure 4–14](#).

**Figure 4–14 Selecting a Horizontal Layout**



You can see in the Visual Editor that the portlets and images are now side by side, rather than on top of one another as seen in [Figure 4–15](#).

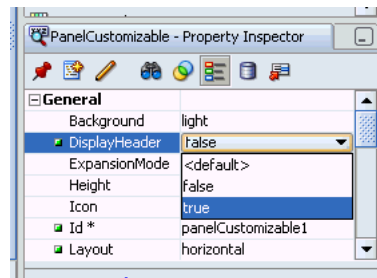
Figure 4–15 Horizontal Layout View



Now let's activate the ability to hide or show content.

- In the Property Inspector, under General, locate **DisplayHeader**, and change it to **true** as shown in Figure 4–16.

Figure 4–16 Changing DisplayHeader to True



This displays a header area at the top of the layout so that there is room for the Action icon to appear as part of the menu. (If for some reason you don't want to display the header in your own applications, you can keep `DisplayHeader` set to `false`. As long as you complete the next step, the menu actions for the component are displayed by hovering over the `panelCustomizable` area.)

- In the Property Inspector, under Actions, locate `IsSeededInteractionAvailable`, and change it to **true**.

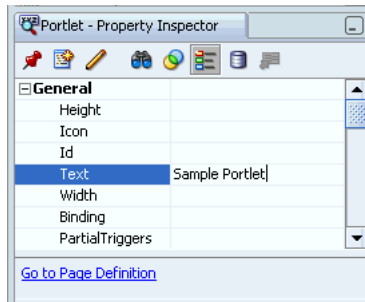
The `IsSeededInteractionAvailable` attribute makes it possible to display whatever menu actions are supported by the component. In this case, the user will be able to hide or show any content that appears within the layout.

Now let's change the name of `MyJavaPortlet` through the Property Inspector, just so you can see how it's done.

Note that once you make changes through the Property Inspector, you can never again customize or personalize the portlet's name at runtime. Although this behavior is acceptable for the purposes of this tutorial, it's something you'll want to keep in mind as you develop your own applications in the future.

- In the Structure window, highlight **adf:portlet** so that the Property Inspector now reflects the portlet's properties.
- In the Property Inspector, under General, locate the Text field, and enter: `Sample Portlet` as shown in Figure 4–17.

This changes the name that appears in the portlet header.

**Figure 4–17 Renaming the Portlet**

10. Click **Save All** in the JDeveloper toolbar.

Let's run the page again to see how these changes affect the page.

## Step 4: Testing the New Customizations

To test the customizations, we run the page and make sure it looks as we expect it to:

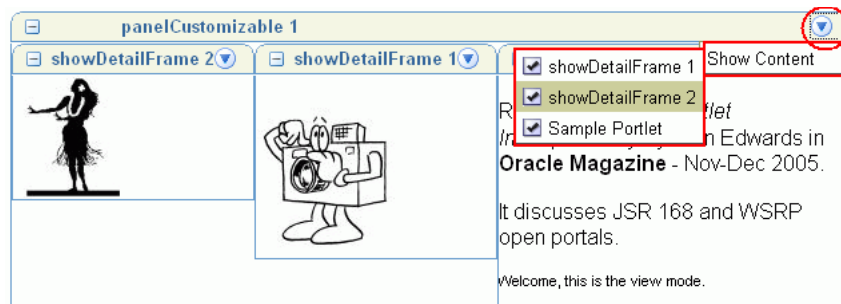
1. Before running the page again, stop the embedded OC4J Server. Choose **Run, Terminate, Embedded OC4J Server** from the main menu.  
Alternatively, click the red square Terminate icon in the Embedded OC4J Server Log window.
2. Click **MyPage.jspx** in the Visual Editor, then click the green arrow in the toolbar to run the page.

Assuming there aren't any compiler errors, the page opens in a new browser window (Figure 4–18).

**Figure 4–18 panelCustomizable with Two showDetailFrames and a Portlet**

3. As a result of activating `IsSeededInteractionAvailable`, the container object's Action icon now contains a **Show Content** action, through which you can hide or show the images and portlet (Figure 4–19). Take a moment now to experiment with them.

Figure 4–19 Show Content Action



All the other page customizations we made in Step 3 are also there:

- The title of the portlet has changed to Sample Portlet. Now that the portlet title is driven by the Text property you can no longer customize or personalize the title at runtime. If you click the Actions icon and choose Customize, you'll see this for yourself.
- The components are displayed horizontally, instead of vertically. If you click the Actions icon on one of the images, you can see that the Move action now says Left and Right instead of Up and Down.
- Because we changed DisplayHeader to true, the container object, panelCustomizable, now displays a header area. Recall that the layout had no such header in Step 3, before we changed the DisplayHeader setting.

Everyone who runs the page will see these page customizations.

4. Before moving on, let's remove the *permanent* portlet title customization we made through the Property Inspector. In [Chapter 8, "Providing Security"](#) we use the portlet's title property to demonstrate page personalization so we must be able to change the name at runtime:
  - a. Close the browser, and return to JDeveloper.
  - b. In the Structure Window for MyPage.jpsx, highlight `adfp:portlet`.
  - c. In the Property Inspector, click the **Text** field, and then click the **Reset to Default** icon in the Property Inspector toolbar. This will remove the text Sample Portlet.

In the Visual Editor, the name in the portlet header should change back to MyJavaPortlet1.

In the future, you may want to use multiple `panelCustomizable` components on a page, each containing one or more portlets or objects within a `showDetailFrame` component. Because each layout has its own Move, Maximize, and Show/Hide capabilities, placing multiple layouts on the page enables you to treat blocks of components and portlets as separate entities. You'll find more information in the *Oracle WebCenter Framework Developer's Guide*.

## Step 5: Changing the Look and Feel

In Oracle WebCenter Framework, there are two ways to control the style (that is, look and feel) of your application:

- Use an Oracle ADF Faces skin to apply a global style to the entire application. If you choose this option, you can use *style selectors* in your own custom skin to modify selected aspects of a component or area of a component.

- Use the Property Inspector to change style-related properties at design time.

In the following steps, you'll use both of these methods to change the look and feel of `MyPage.jspx`:

- [Step 5a: Changing showDetailFrame Background Using the Default ADF Faces Skin](#)
- [Step 5b: Using Custom Skins to Apply Your Own Styles](#)

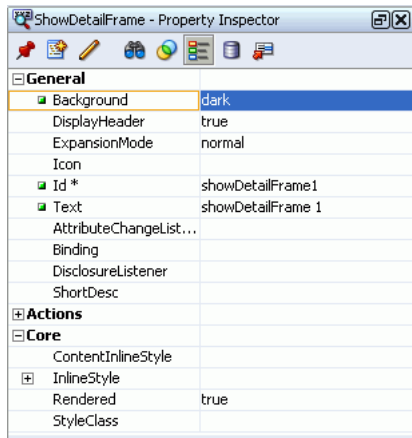
## Step 5a: Changing showDetailFrame Background Using the Default ADF Faces Skin

In this step, you'll use the default ADF Faces skin (called *Oracle*) to influence style-related properties.

1. In the Structure window, highlight one of the `cust:showDetailFrame` entries. We'll use this component to see how to specify style property values.
2. In the Property Inspector, expand the **Background** property.
 

The default ADF Faces skin offers three settings, light, medium, and dark. Let's choose a dark background for this component.
3. Change the value to **dark**, as shown in [Figure 4–20](#).

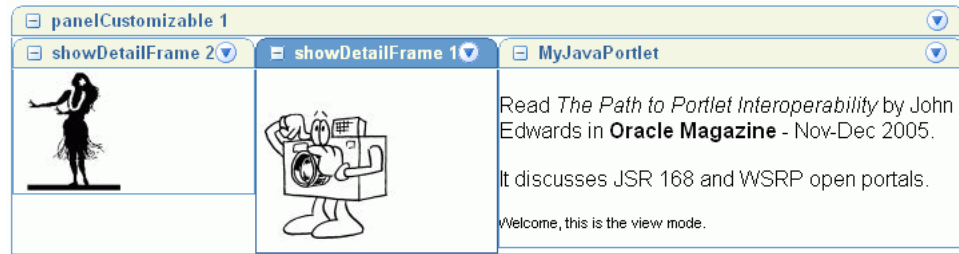
**Figure 4–20 Property Inspector: Background**



4. Click **File, Save All** to save your work.
5. Click the red square to stop the embedded OC4J in preparation to run your page again.
6. Right-click **MyPage.jspx**, and click **Run**.

Assuming there aren't any compiler errors, the page opens in a new browser window ([Figure 4–21](#)). Notice how the background decoration of the `showDetailFrame` has changed.

Figure 4–21 Image with Dark Background



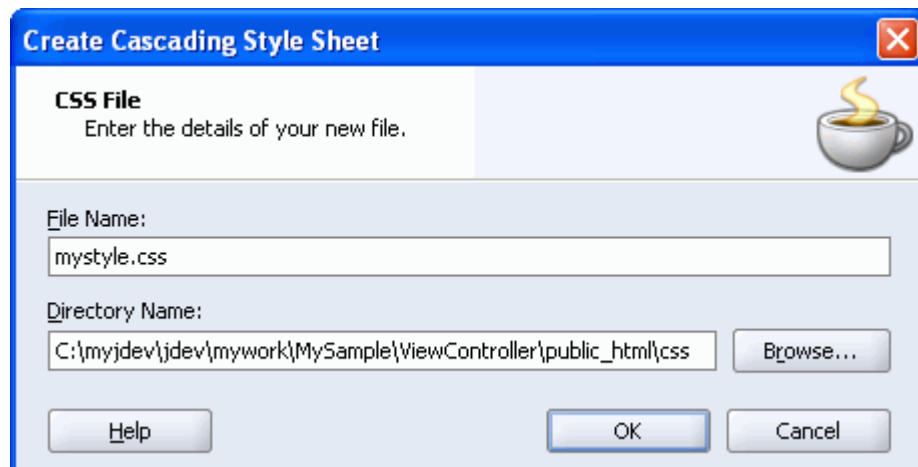
Now let's see how to create a brand new skin and override the styles defined by the default ADF Faces skin.

## Step 5b: Using Custom Skins to Apply Your Own Styles

In this step you'll create a new skin, register the skin with your application, and learn how to apply custom styles to components on MyPage.

1. First, return to JDeveloper and create a brand new style sheet (.css):
  - a. In the Applications Navigator, right-click **ViewController**, and choose **New**.
  - b. In the New Gallery under Categories, expand the Web Tier, and select **HTML**.
  - c. Under Items, select **CSS File**, and click **OK**.
  - d. For File Name, enter `mystyle.css` as shown in [Figure 4–22](#).

Figure 4–22 Create Cascading Style Sheet



- e. Click **OK**.

Your new style sheet appears in the Applications Navigator under ViewController, Web Content, css and also displays in the Editor. Notice the default style selectors BODY, H1, H2, and H3.

In the steps that follow, you'll register a new *skin* with your application. This involves creating a file named `adf-faces-skins.xml`, and populating it with a short list of tags that identify the skin's ID, location, .css, and the like.

2. Let's create `adf-faces-skins.xml`:

- a. In the Applications Navigator, under ViewController, Web Content, right click the **WEB-INF** folder, and select **New**.
- b. In the New Gallery, set the Filter By scope to **All Technologies**.
- c. Under the General node, select **XML**.
- d. In the right pane, select **XML Document**, and click **OK**.
- e. In the File Name field, enter the file name: `adf-faces-skins.xml` as shown in [Figure 4-23](#).

**Figure 4-23** Create `adf-faces-skins.xml`



- f. The file must be stored in the WEB-INF folder, so just click **OK** to create the file.  
An empty XML file displays in the Editor. It appears in the Applications Navigator under WEB-INF.
3. Add tags to `adf-faces-skins.xml` to identify your new skin (and point to your new style sheet `mystyle.css`):

- a. Copy and paste the following code into the XML Editor:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<skins xmlns="http://xmlns.oracle.com/adf/view/faces/skin">
  <skin>
    <id>mystyle</id>
    <family>mystyle</family>
    <render-kit-id>oracle.adf.desktop</render-kit-id>
    <style-sheet-name>css/mystyle.css</style-sheet-name>
  </skin>
</skins>
```

Your file should look [Figure 4-24](#).



**Figure 4–24** Configure *adf-faces-skins.xml*


```

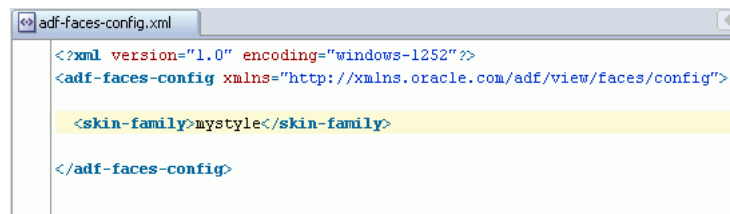
<?xml version="1.0" encoding="ISO-8859-1"?>
<skins xmlns="http://xmlns.oracle.com/adf/view/faces/skin">
  <skin>
    <id>mystyle</id>
    <family>mystyle</family>
    <render-kit-id>oracle.adf.desktop</render-kit-id>
    <style-sheet-name>css/mystyle.css</style-sheet-name>
  </skin>
</skins>

```

- b. Click **File, Save** to save *adf-faces-skins.xml*.

In the next step you'll configure your application to use this new skin by setting the `<skin-family>` tag in *adf-faces-config.xml*.

4. Edit the `<skin-family>` tag in *adf-faces-config.xml*:
  - a. Open *adf-faces-config.xml*, located under ViewController, Web Content, WEB-INF.
  - b. Replace `oracle` (the default skin) with the family name of your new skin. Enter `mystyle` as shown in [Figure 4–25](#).

**Figure 4–25** Edit Skin Family in *adf-faces-config.xml*


```

<?xml version="1.0" encoding="windows-1252"?>
<adf-faces-config xmlns="http://xmlns.oracle.com/adf/view/faces/config">
  <skin-family>mystyle</skin-family>
</adf-faces-config>

```

- c. Click **File, Save**.

Now the new skin is registered with your application, let's add some style selectors to your style sheet that apply to `showDetailFrame` components on `MyPage`.

5. Add the following style selectors to *mystyle.css*:
  - a. In the Applications Navigator, under ViewController, Web Content, *css*, double-click **mystyle.css**.
  - b. Copy the following code, and paste it at the bottom of the file:

```

af|showDetailFrame::header-light
{
  color:Purple;
}
af|showDetailFrame::header-medium
{
  color:Purple;
}
af|showDetailFrame::header-dark
{
  color:White;
}

```

This code specifies the color for the text that displays in a `showDetailFrame` component's header and a portlet's header. We've chosen *purple* text for header-light and header-medium, and *white* text for header-dark. Choose another color scheme if you prefer.

- c. Copy the following code, and paste it at the bottom of the file:

```
af|showDetailFrame::content-light, af|showDetailFrame::content-medium,
af|showDetailFrame::content-dark
{
  color: Black;
  background-color: Silver;
  border-left:1px black solid;
  border-right:1px black solid;
  border-bottom:1px black solid;
}
```

This code changes the content of a `showDetailFrame` and portlet body to have a silver background, with black text and black borders. Choose another color scheme if you prefer.

For a full list of style selectors for `showDetailFrame` and `panelCustomizable` components, see *Oracle WebCenter Framework Developer's Guide*.

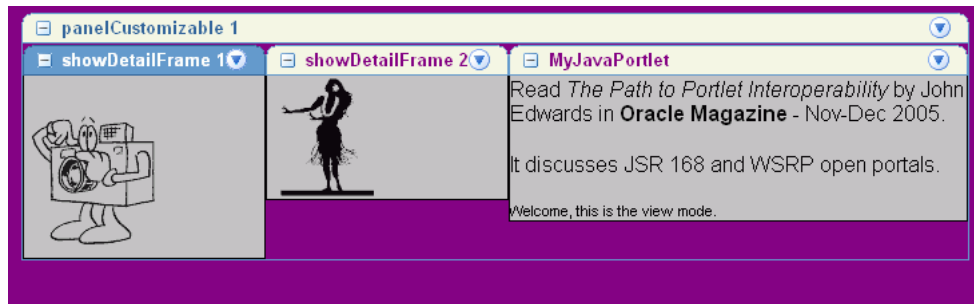
6. Now change the background color of your page. Scroll to the `BODY` tag at the top of the style sheet, and change the `background-color` from `White` to `Purple`. The `BODY` tag should look like this:

```
BODY
{
  background-color: Purple;
  color: black;
  font-family: Arial, Helvetica, sans-serif;
}
```

7. Click **File, Save All** to save your work.
8. Click the red square to stop the embedded OC4J in preparation to run your page again.
9. Right-click **MyPage.jspx**, and click **Run**.

Assuming there aren't any compiler errors, the page opens in a new browser window.

Take a moment to examine the new look and feel. If you used the styles we suggested your page will look like [Figure 4-26](#). Notice how `showDetailFrames` display purple/white header text, black content text, and have a silver background. The background of the page will be colored purple too.

**Figure 4–26** *MyPage.jspx Using mystyle.css*

During this lesson you've experimented with ways to affect the look and feel of your application. You'll find more detailed information in the *Oracle WebCenter Framework Developer's Guide*.

Before moving on, let's change back to the default Oracle skin.

10. To remove the custom skin:
  - a. Open `adf-faces-config.xml`, located under `ViewController, Web Content, WEB-INF`.
  - b. Change the `<skin-family>` tag back to `oracle` as shown in [Figure 4–27](#).

**Figure 4–27** *Edit Skin Family in adf-faces-config.xml*

- c. Click **File, Save All**.

In the next lesson, you'll learn how to add the Rich Text portlet, which provides a bulletin board-like feature to users.



---

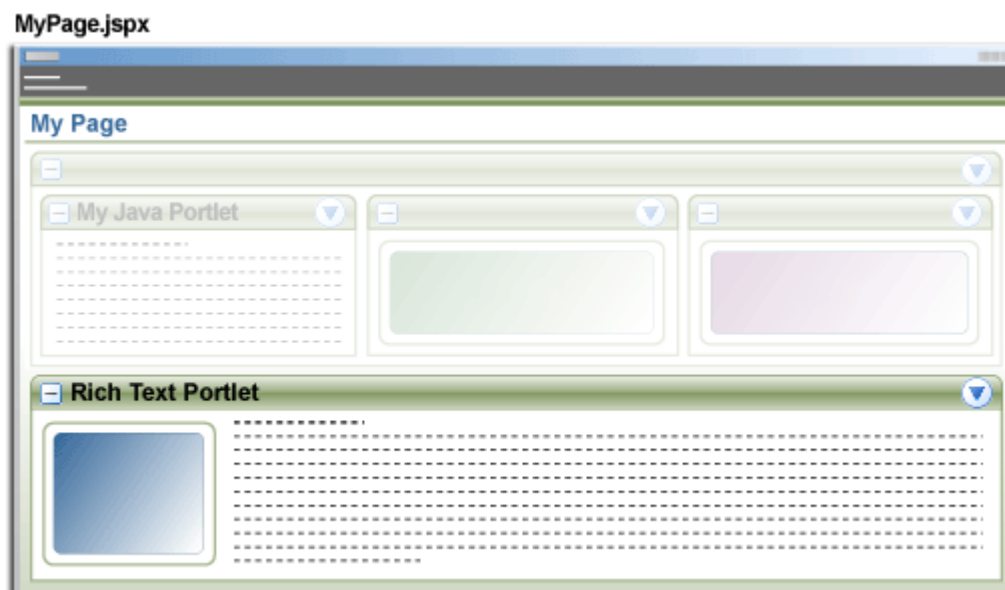
## Adding the Rich Text Portlet

The Rich Text portlet is a useful tool for posting enterprise announcements and news items at runtime. Once you add the portlet to a page, authorized users can use the Actions icon to invoke a toolbar with all the rich-text editing tools needed to insert, update, and format display text. Depending on the security privileges set for the page, the Rich Text portlet can be used to broadcast information to a wide audience, or to a more narrowly defined group.

The Rich Text portlet is available through the WebCenter Preconfigured OC4J.

Figure 5-1 shows what your page will look like at the end of the lesson.

**Figure 5-1** *MyPage.jspx at the End of Lesson 5*



### Introduction

This lesson shows you how to work with the Rich Text portlet through the following steps:

- [Step 1: Registering the Rich Text Producer](#)
- [Step 2: Adding the Rich Text Portlet To Your Page](#)
- [Step 3: Customizing the Rich Text Portlet at Run Time](#)

## Prerequisites

The Rich Text portlet is available through the WebCenter Preconfigured OC4J (installed and initialized during [Chapter 3 Step 2: Setting Up Your Connection](#)). If this OC4J instance is no longer running, click the **Start WebCenter Preconfigured OC4J** icon at the far right of the JDeveloper toolbar before you begin this lesson.

## Step 1: Registering the Rich Text Producer

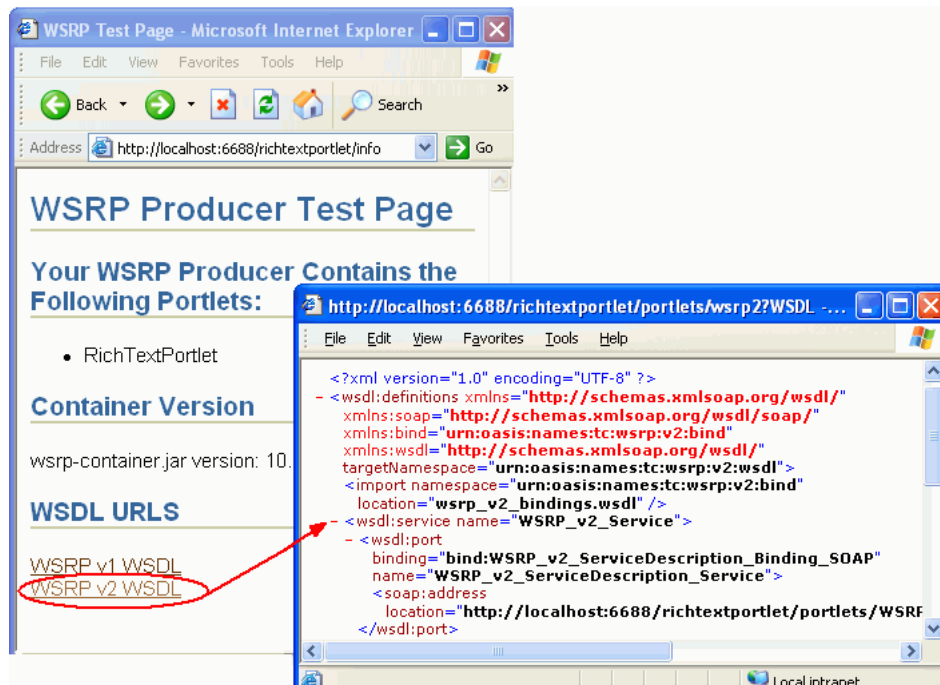
In this step you'll register the Rich Text producer with your application. But first, let's check that the Rich Text producer is available through the WebCenter Preconfigured OC4J:

1. In a browser, enter the Rich Text producer test page URL:

`http://localhost:6688/richtextportlet/info`

If the producer is up and running you should see the WSRP Producer Test Page shown in [Figure 5-2](#).

**Figure 5-2** Rich Text Producer Test Page



The test page displays two Rich Text producer registration URLs (WSDL documents). You'll use the **WSRP v2 WSDL** URL in this WebCenter application:

`http://localhost:6688/richtextportlet/portlets/wsrp2?WSDL`

**Tip:** Test pages for *every* portlet producer available through the WebCenter Preconfigured OC4J are accessible from:  
`http://localhost:6688/`

Now, let's register the Rich Text producer.

2. In the Applications Navigator, right-click **Portlet Producers**, then click **New WSRP Producer** as shown in [Figure 5-3](#).

**Figure 5–3 Registering the Rich Text Portlet with the Application**

Because you already have one portlet producer registered, you can use this right-click shortcut rather than having to navigate through the New... gallery.

3. Click **Next** to exit the wizard's Welcome screen.
4. In the Name field, enter: `RichTextProducer`
5. Click **Next**.
6. In the URL Endpoint field, enter:

`http://localhost:6688/richtextportlet/portlets/wsrp2?WSDL`

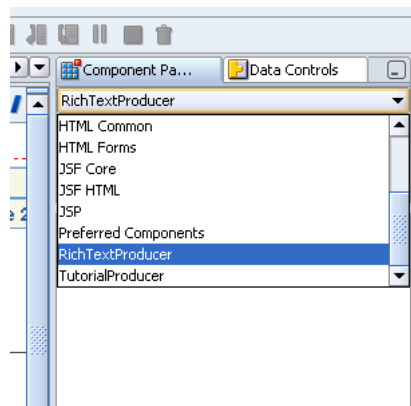
This tutorial assumes that `localhost` can be used in a URL to successfully refer to the local computer on which your preconfigured OC4J installation resides. (If this is not the case, replace `localhost` with your computer's IP address.)

7. Click **Next**.
8. Click **Finish** to exit the wizard.
9. Click **OK** to close the message box.

Now you are ready to drop the Rich Text portlet onto your page.

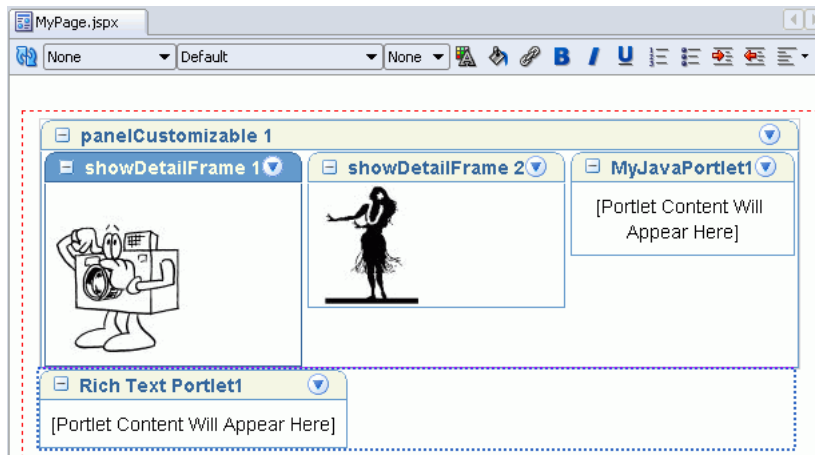
## Step 2: Adding the Rich Text Portlet To Your Page

1. Display `My Page.jspx` in the Visual Editor.
2. From the Component Palette, select **RichTextProducer** (Figure 5–4).

**Figure 5–4 Rich Text Portlet Producer**

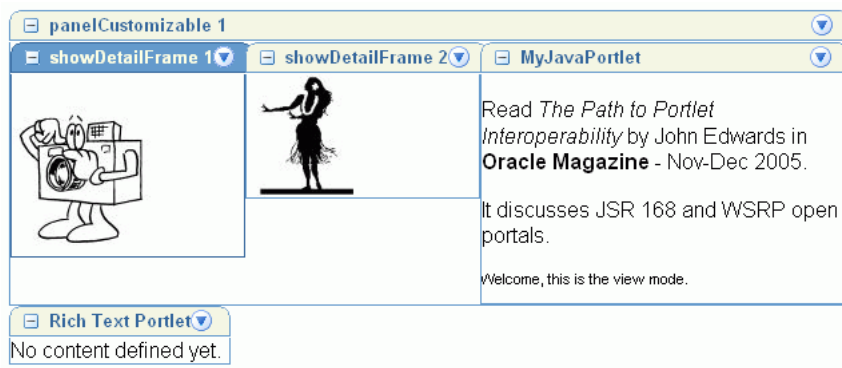
3. Select **Rich Text Portlet**, and drop it on top of `h: form` in the Structure window as shown in [Figure 5-5](#).

**Figure 5-5 Adding the Rich Text Portlet to the Page**



4. Select **Rich Text Portlet1** in the Visual Editor and you'll see its properties displayed in the Property Inspector.
5. In the Property Inspector, under Display Options, set `AllModesSharedScreen` to **true**.  
This directs the page to render all changes made through personalize/customize inline, rather than by refreshing the entire page.
6. Select **File, Save All** to save your work.
7. Select **Run, Run MyPage.jspx**, or click the green arrow in the JDeveloper toolbar.  
A new browser opens to display the page ([Figure 5-6](#)).

**Figure 5-6 Default Rich Text Portlet**



This is what the user sees when the page is displayed. The next step shows you how easy it is to customize the Rich Text portlet, and may suggest some uses as you plan your WebCenter application.



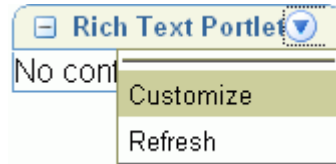
## Step 3: Customizing the Rich Text Portlet at Run Time

Once you've placed the Rich Text portlet on a page, users can use the Actions icon to enter text that will be visible to everyone.

1. Click the **Actions** icon in the Rich Text portlet's header.

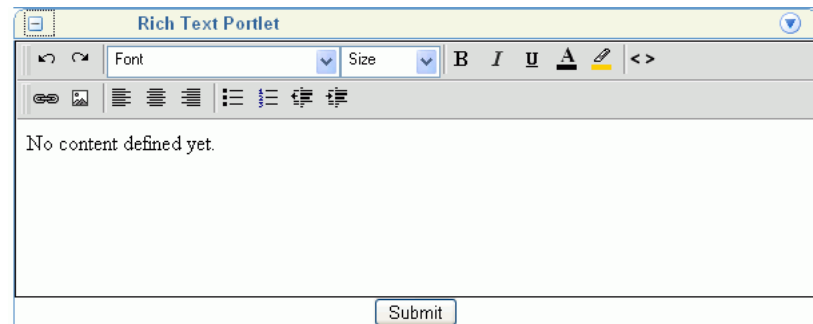
The Rich Text portlet should now look something like [Figure 5-7](#).

**Figure 5-7** The Rich Text Portlet's Action Menu



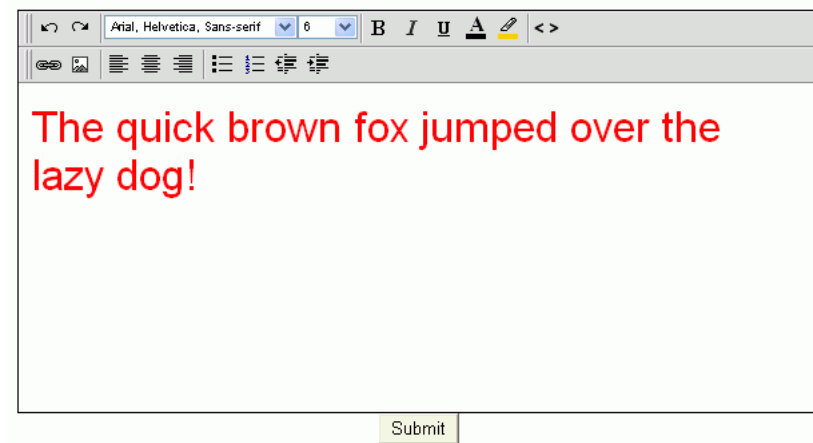
2. Click **Customize** to display the rich text controls ([Figure 5-8](#)).

**Figure 5-8** The Rich Text Portlet's Text Controls



3. Enter some text of your own, and change the font size, type, and color, and choosing a background color. For example, see [Figure 5-9](#).

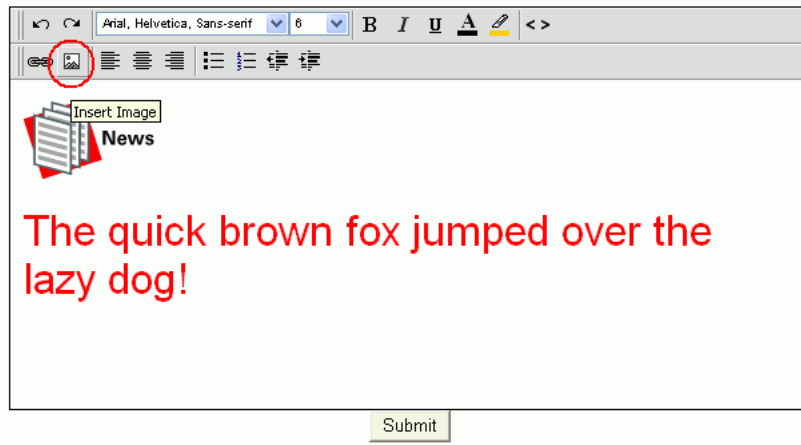
**Figure 5-9** Text in Arial Font, Size 6, in Red



4. Experiment with the other controls. For example, display an image in the portlet:
  - a. Click the **Insert Image** icon.

- b. Enter the URL of any image, and click **OK** (Figure 5–10).

**Figure 5–10** Insert Image Using the Rich Text Editor



5. When you have finished, click **Submit**.

The page is refreshed to display the new text and image.

For a complete description of the Rich Text portlet, as well as the other pre-packaged portlets, see the *Oracle WebCenter Framework Developer's Guide*.

Now let's move on to see how to enable communication between two portlets—in effect, how to pass a parameter from one portlet to drive the content that is displayed in the other.

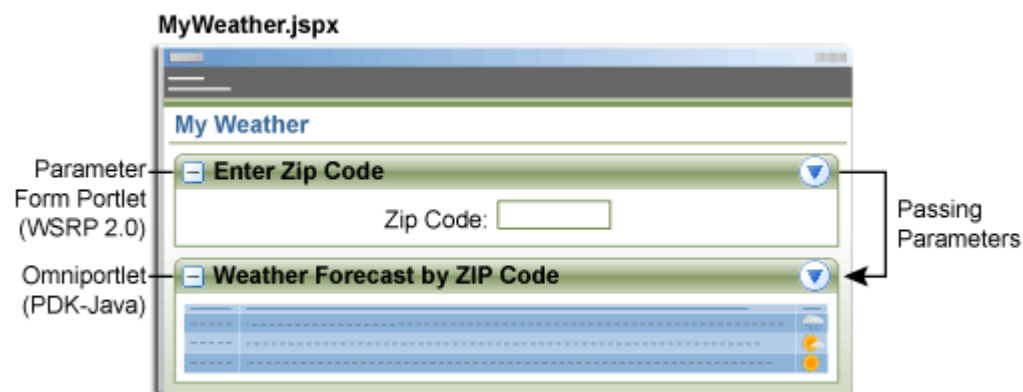
---

## Making Portlets Communicate

In this lesson you'll add two portlets to a page and learn how to configure them so that one portlet drives the content of the other. To demonstrate inter-portlet communication, you'll use a Parameter Form Portlet (one of the sample WSRP 2.0 portlets) and an instance of OmniPortlet (an Oracle PDK portlet). Both portlets are available through the preconfigured OC4J you installed in [Chapter 3, "Building and Testing Your First Portlet"](#).

[Figure 6-1](#) illustrates what the page (called `MyWeather.jspx`) will look like at the end of the lesson.

**Figure 6-1** *MyWeather.jspx at the End of Lesson 6*



### Introduction

You'll make two portlets communicate in the following steps:

- [Step 1: Registering Portlet Producers](#)
- [Step 2: Placing the Parameter Form Portlet on a Page](#)
- [Step 3: Customizing the Parameter Form Portlet](#)
- [Step 4: Placing an OmniPortlet on a Page](#)
- [Step 5: Building an OmniPortlet That Uses a Web Service](#)
- [Step 6: Configuring the Portlets Together](#)
- [Step 7: Testing Portlet Interaction](#)

## Prerequisites

Both of the portlets we use in this lesson are available through the WebCenter Preconfigured OC4J (installed and initialized during [Chapter 3 Step 2: Setting Up Your Connection](#)). If this OC4J instance is no longer running, click the **Start WebCenter Preconfigured OC4J** icon at the far right of the JDeveloper toolbar before you begin this lesson.

## Step 1: Registering Portlet Producers

Before you can add a portlet to a page, you must register its producer with the WebCenter application. For this lesson you'll need to register producers for the following portlets:

- Parameter Form Portlet (WSRP portlet producer)
- OmniPortlet (Oracle PDK portlet producer)

Let's start with the Parameter Form Portlet producer:

1. Check that the Parameter Form Portlet producer is up and running. In a browser, enter the following producer test page URL:

```
http://localhost:6688/portletapp/info
```

If the producer is up and running, you should see a WSRP Producer Test Page displaying two WSRP producer registration URLs (WSDL documents). You'll use the **WSRP v2 WSDL** URL in this WebCenter application:

```
http://localhost:6688/portletapp/portlets/wsrp2?WSDL
```

Now, let's register the producer with your application.

2. Right-click **Portlet Producers** in the Applications Navigator, and click **New WSRP Producer**.
3. When the WSRP Portlet Producer Registration Wizard displays, click **Next** to move beyond the Welcome page.
4. In the Name field, enter: `SampleWSRPPortletsProducer`
5. Click **Next**.
6. On the Connection page, enter the producer's **URL Endpoint**.

To produce the Parameter Form portlet through the WebCenter Preconfigured OC4J, enter the following URL:

```
http://localhost:6688/portletapp/portlets/wsrp2?WSDL
```

This tutorial assumes that `localhost` can be used in a URL to refer to the local computer on which your preconfigured OC4J installation resides. (If this is not the case, replace `localhost` with your computer's IP address.)

Proxy information isn't necessary as we're using a local preconfigured OC4J. Make sure that the **Use proxy** option is unchecked.

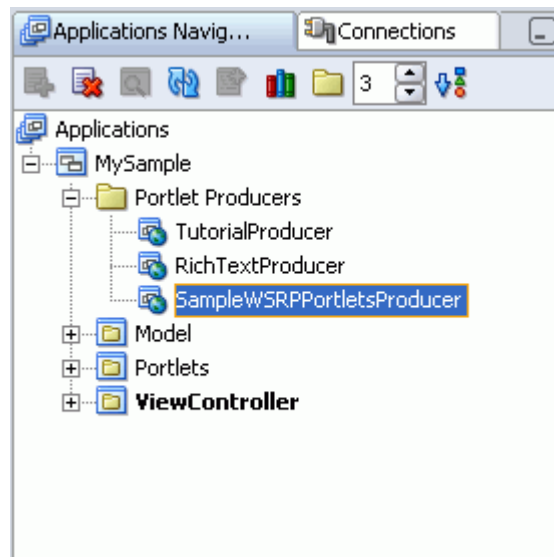
7. Click **Next**.
8. On the Registration Details page, accept the default timeout value of 30 seconds, and click **Finish**. For this tutorial, there is no need to set any of the advanced options.

You should see the following message ([Figure 6-2](#)):

**Figure 6–2 WSRP Portlet Producer Registration Successful**

9. Click **OK** to close the message box.

The new producer appears in the Applications Navigator, under the section Portlet Producers (Figure 6–3).

**Figure 6–3 Successfully Registered Producer for the Sample WSRP Portlets**

Open the Component Palette through the **View** menu, and expand its drop-down list. Locate and select **SampleWSRPPortletsProducer** from this list. Several portlets are available for selection. Among these, you will find the Parameter Form Portlet, which you will use in this lesson. (If you can't see anything in the Component Palette, try displaying a Java Server Faces page in Design view, such as MyPage.jspx. This should reveal all the portlets).

You'll come back to the Parameter Form Portlet later. First you must register the OmniPortlet producer.

10. First, check that the OmniPortlet producer is up and running. In a browser, enter the following producer test page URL:

```
http://localhost:6688/portalTools/omniPortlet/providers/omniPortlet
```

If the producer is available, you'll see the OmniPortlet's Producer Test Page.

Now, let's register the OmniPortlet producer. So far in this tutorial you've registered only WSRP producers. As OmniPortlet is an Oracle PDK portlet, you'll be using a different wizard to register this producer.

11. Right-click **Portlet Producers** in the Applications Navigator, and click **New Oracle PDK Producer**.

12. When the PDK Portlet Producer Registration Wizard appears, click **Next**.

13. In the Name field, enter: `OmniPortletProducer`

14. Click **Next**.

15. On the Connection page, enter the producer's **URL Endpoint**.

To produce OmniPortlet through the WebCenter Preconfigured OC4J, enter the following URL:

```
http://localhost:6688/portalTools/omniPortlet/providers/omniPortlet
```

This tutorial assumes that `localhost` can be used in a URL to refer to the local computer on which your preconfigured OC4J installation resides. (If this is not the case, replace `localhost` with your computer's IP address.)

Make sure that the **Use proxy** option is unchecked.

16. Click **Finish**.

You should see the following message ([Figure 6-4](#)):

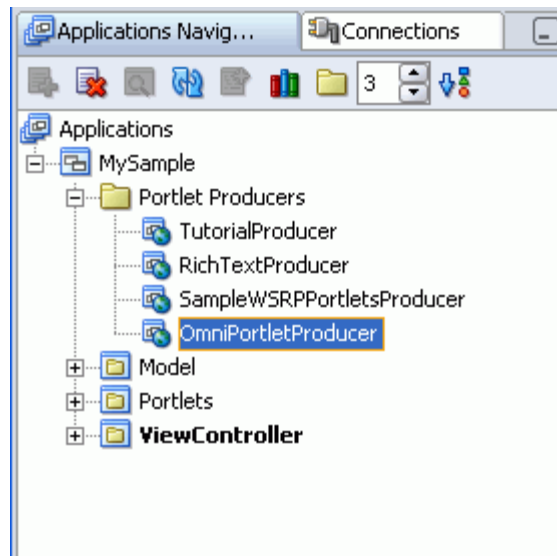
**Figure 6-4 Oracle PDK Portlet Producer Registration Successful**



17. Click **OK** to close the message box.

Both producers should now appear in the Applications Navigator, under the section Portlet Producers ([Figure 6-5](#)).

Figure 6-5 Successfully Registered Producer for OmniPortlet



18. Open the Component Palette again, and click the drop-down list to make sure that **OmniPortletProducer** is displayed. Select **OmniPortletProducer** from this list.

Several portlets are available for selection. Among these, you'll find *OmniPortlet*, which you will use in this lesson.

Now both producers are registered with the tutorial application you can drop their portlets onto a page.

## Step 2: Placing the Parameter Form Portlet on a Page

In this step, you'll place a Parameter Form Portlet on a page and see how JDeveloper handles a portlet with public portlet parameters. Then you'll run the page and, through portlet customization, create a portlet that invites a user to enter a ZIP code. Later on in the lesson you'll learn how to pass the ZIP code as a parameter to an *OmniPortlet*, where it'll be used to drive the content of a weather portlet.

Let's start by creating a brand new page called `MyWeather.jspx`, which you'll use to learn about inter-portlet communication.

1. Create a new Java Server Faces page named `MyWeather.jspx`.
  - a. In the Applications Navigator, right-click **ViewController**, and select **New**.
  - b. In the Categories pane, under Web Tier, select **JSF**.
  - c. Under Items, select **JSF JSP**, and click **OK**.
  - d. Click **Next** to skip the Welcome page.
  - e. In the File Name field, enter: `MyWeather`
  - f. Under Type, click **JSP Document**, and then click **Next**.
  - g. Choose **Automatically Expose UI Components in a New Managed Bean**, and then click **Next**.
  - h. On the Tag Libraries page, make sure the following libraries appear in the **Selected Libraries** pane:

`ADF Faces Components 10_1_3_2_0`

ADF Faces HTML 10\_1\_3\_2\_0

ADF Portlet Components 10\_1\_3\_2\_0

Customizable Components Core 10\_1\_3\_2

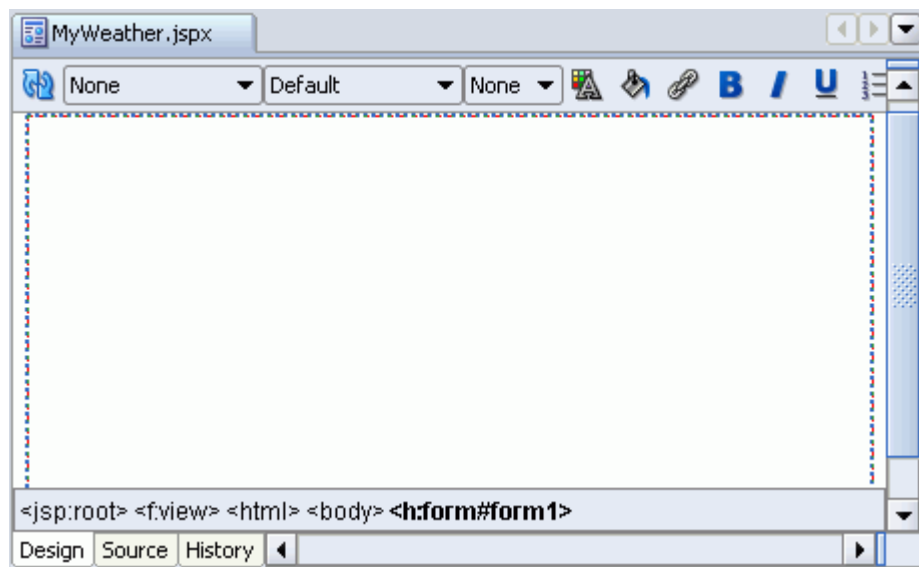
JSF Core 1.0

JSF HTML 1.0

- i. There is no need to set any other options, so click **Finish** on this page.

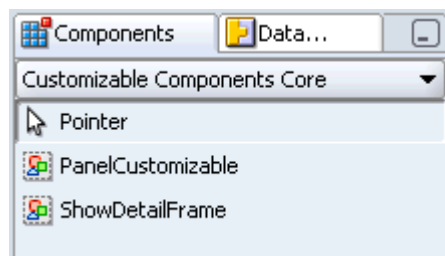
MyWeather.jspx should open in the Visual Editor (Figure 6–6), ready for us to begin adding portlets.

**Figure 6–6** Empty MyWeather.jspx



- 2. Add a `panelCustomizable` component to your page. You'll use this component to house both of your portlets (Parameter Form Portlet and OmniPortlet) and to provide horizontal/vertical layout.
  - a. Open the Component Palette.
  - b. Scroll to the listing for **Customizable Components Core** (Figure 6–7).

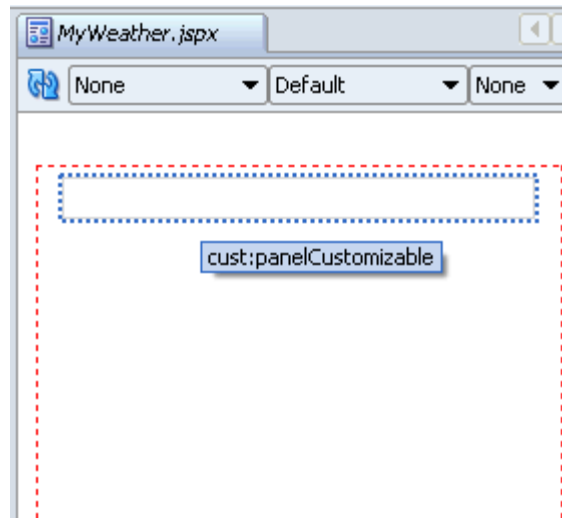
**Figure 6–7** Component Palette - Customizable Components Core



- c. Select **PanelCustomizable**, and drag it over the `h:form` tag in the Structure window.

The `panelCustomizable` component should display at the top of the page (Figure 6–8).



**Figure 6–8** *MyWeather.jspx - PanelCustomizable Component*

The `panelCustomizable` component must be placed inside the `h:form` for the page. Take a look in the Structure window to check that the frame is placed *inside* the form. Remember, you don't need add a `showDetailFrame` because portlets come automatically equipped with the customization capabilities; that is, the ability to maximize/minimize and to reposition the portlet on the page.

3. Now you're ready for the Parameter Form Portlet. From the Component Palette, select **SampleWSRPPortletsProducer**.
4. Select **Parameter Form Portlet**, then drag and drop it on top of the `cust:panelCustomizable` component in the Structure window.

The Visual Editor should look like [Figure 6–9](#).

**Figure 6–9** *MyWeather.jspx - Parameter Form Portlet Inside a panelCustomizable*

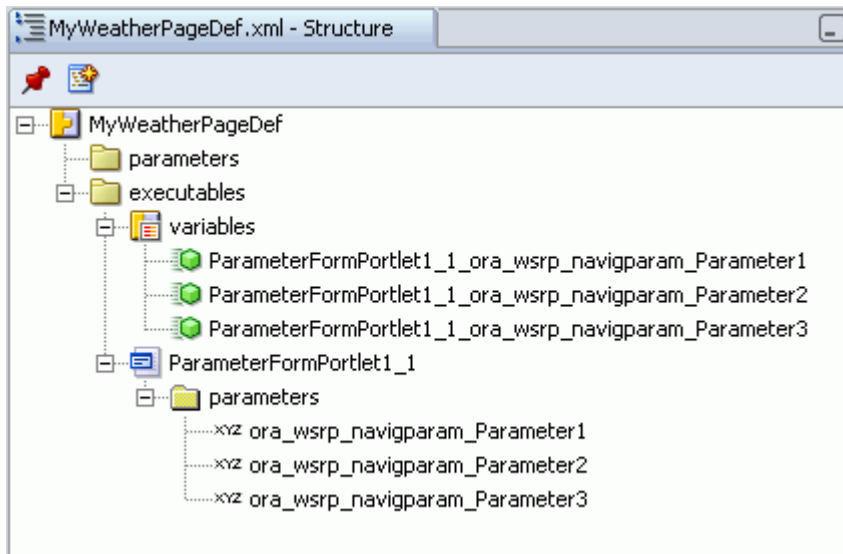
Before you run the page, let's take a look at the underlying page definition to see how JDeveloper responds to a portlet with built-in parameters like this one.

Right-click **MyWeather.jspx** in the Applications Navigator, and choose **Go to Page Definition**. Click **Yes** if the Page Definition doesn't exist yet.

In the Structure window ([Figure 6–10](#)) you should see an entry for each of the Parameter Form's *portlet parameters* (`ora_wsrp_navigparam_Parameter1`, `ora_wsrp_navigparam_Parameter2` and `ora_wsrp_navigparam_Parameter3`). You should also see three *page variable* entries with corresponding

names and the prefix <portletname\_n\_> —in this case, ParameterFormPortlet1\_1\_).

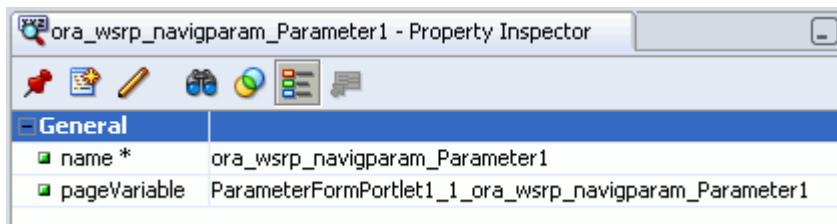
**Figure 6–10 MyWeatherPageDef.xml - Parameter Form Portlet Parameters**



When a portlet is dropped on a page, JDeveloper checks to see if the portlet has public portlet parameters and, if it does, automatically adds the corresponding number of page variables and maps them to the portlet parameters.

Use the Property Inspector to examine one of the mappings. You should see that `ora_wsrp_navigparam_Parameter1`, for example, is mapped to the page variable named `ParameterFormPortlet1_1_orawsrp_navigparam_Parameter1` (Figure 6–11).

**Figure 6–11 Property Inspector - Portlet Parameter Mapped to Page Variable**



Now let's run the page to see what a default Parameter Form Portlet looks like.

5. First, save all your changes. Click the **Save All** icon on the JDeveloper toolbar.
6. In the Applications Navigator, right-click **MyWeather.jspx**, and choose **Run**. The default Parameter Form Portlet displays in your browser (Figure 6–12).

**Figure 6–12 Default Parameter Form Portlet**

As you can see, the initial version of the portlet displays some default text. In the next step, you'll customize the portlet so that it invites users to enter a US ZIP code. Remember, the purpose of this lesson is to work with this portlet so it will accept a parameter value (such as a ZIP code) and pass it to another portlet.

## Step 3: Customizing the Parameter Form Portlet

In this step you'll customize the Parameter Form Portlet by changing its title and prompts. You'll also hide unwanted parameters. The Parameter Form Portlet provides three parameter but this tutorial requires only one to demonstrate parameter passing between portlets.

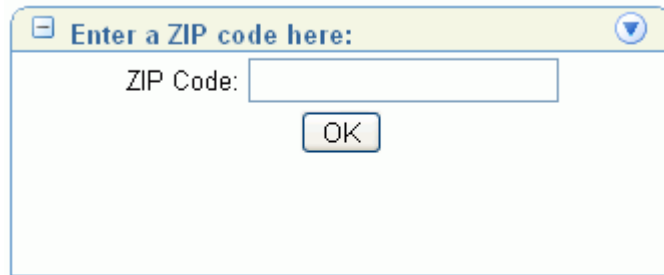
1. Display the portlet in a browser, then click the **Actions** icon in the portlet header.
2. Click the menu option **Customize**.
3. Change the Title to: Enter a ZIP code here:
4. For Parameter 1 Prompt, enter: ZIP Code:
5. You don't need the second and third parameters, so you can hide them. To do this, remove the default prompt text taking care not to leave behind any spaces.

The Customize options should now look something like [Figure 6–13](#).

**Figure 6–13 Parameter Form Portlet Customization Options**

6. Click **OK**.

When displayed in the browser, the portlet should look like [Figure 6–14](#).

**Figure 6–14 Customized Parameter Form Portlet**

Before moving on to the next step, let's review what you've built so far. First, you created a page called `MyWeather.jspx`, and added a single Parameter Form Portlet. Then, through portlet customization, you set up one of its parameters, `ora_wsrp_navigparam_Parameter1`, to accept a US ZIP code. You may recall that JDeveloper automatically mapped this particular parameter to a page variable named `ParameterFormPortlet1_1_ora_wsrp_navigparam_Parameter1`. Later on, you'll use this variable to pass the ZIP code on to another portlet.

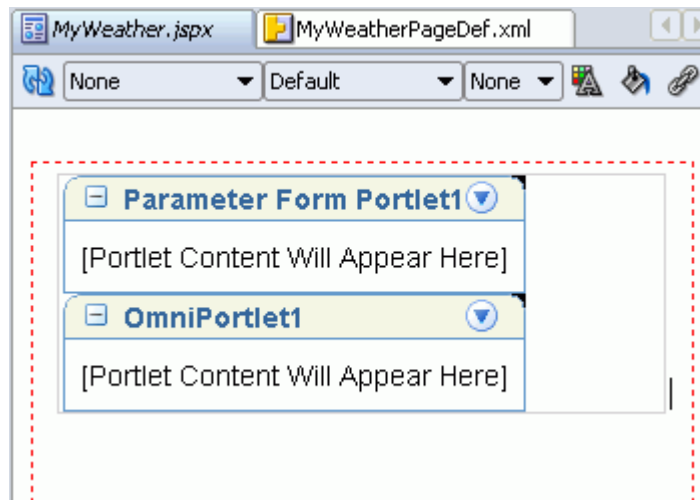
But first, let's add the second portlet—an OmniPortlet.

## Step 4: Placing an OmniPortlet on a Page

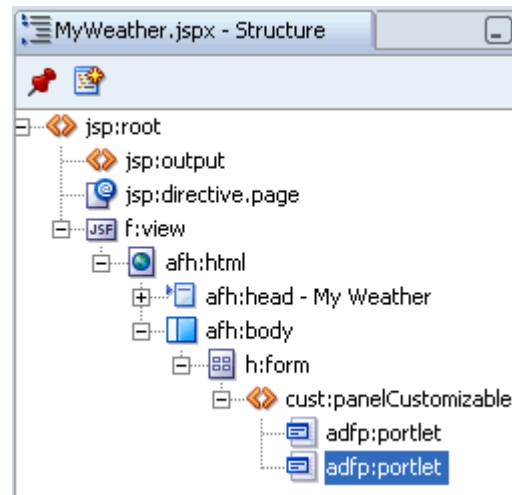
In this step you'll place an OmniPortlet under the Parameter Form Portlet.

1. Display `MyWeather.jspx` in JDeveloper. In the Applications Navigator, right-click `MyWeather.jspx`, and click **Open**.
2. Open the Component Palette, and locate **OmniPortletProducer**.
3. Select **OmniPortletProducer**.
4. Select **OmniPortlet**, and drag it over `cust:PanelCustomizable` in the Structure window.

The Visual Editor should now look like [Figure 6–15](#).

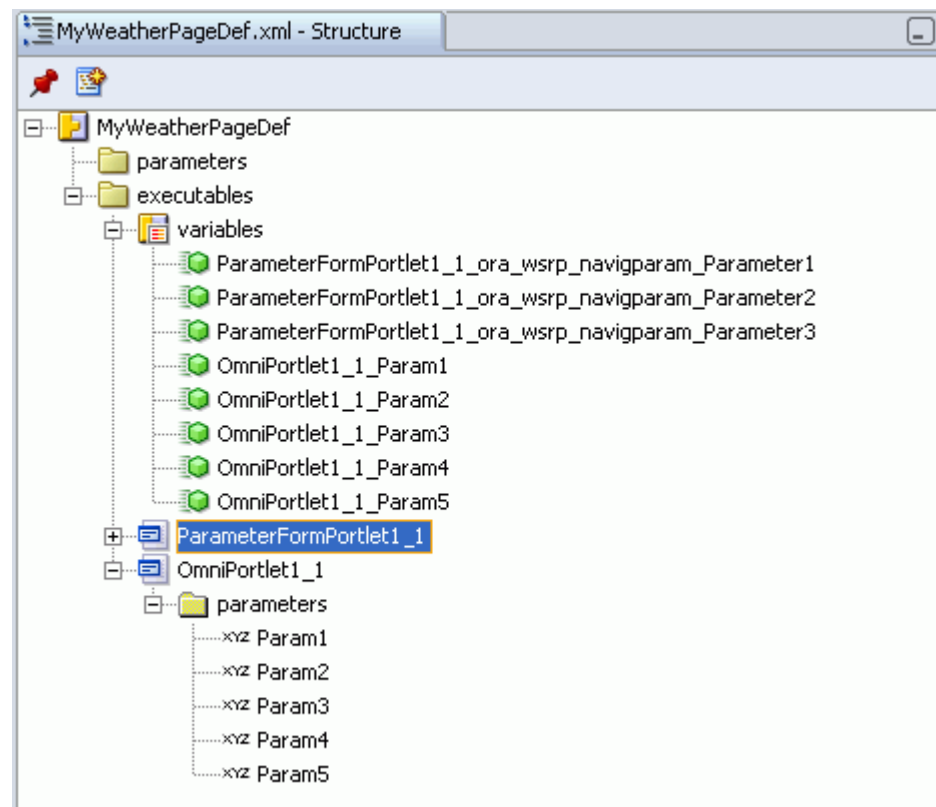
**Figure 6–15 MyWeather.jspx - OmniPortlet Added To PanelCustomizable**

The Structure window should look like [Figure 6–16](#).

**Figure 6–16 Structure Window - Displays New OmniPortlet**

- Let's take another look at the underlying page definition. Right-click **MyWeather.jspx** in the Applications Navigator, and choose **Go to Page Definition**.

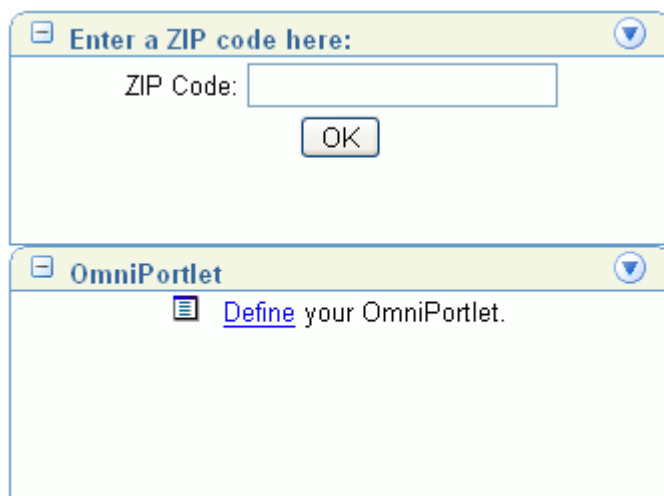
In the Structure window (Figure 6–17), you'll see a new entry for OmniPortlet. This portlet has five *portlet parameters* (Param1 - Param5), and they too have been automatically mapped to *page variables* OmniPortlet1\_1\_Param1 - OmniPortlet1\_1\_Param5.

**Figure 6–17 MyWeatherPageDef.xml - OmniPortlet Portlet Parameters**

In [Step 6: Configuring the Portlets Together](#) you'll use these portlet parameters to configure inter-portlet communication. But first, let's see what an OmniPortlet portlet looks like.

6. First, save all your changes. Click the **Save All** icon on the JDeveloper toolbar.
7. Before running the page again, stop the embedded OC4J Server. Choose **Run, Terminate, Embedded OC4J Server** from the main menu.  
Alternatively, click the red square Terminate icon in the Embedded OC4J Server Log window.
8. In the Applications Navigator, right-click **MyWeather.jspx**, and choose **Run**.  
OmniPortlet displays in the browser underneath the Parameter Form Portlet ([Figure 6–18](#)).

**Figure 6–18** Default (Blank) OmniPortlet



As you can see, the initial version of the portlet is blank. In the next step you'll use OmniPortlet to build a weather portlet based on a Web Service.

## Step 5: Building an OmniPortlet That Uses a Web Service

As one of the most versatile portlets, OmniPortlet can publish data from a variety of data sources using a variety of layouts. In this step you will build a weather portlet based on a Web Service.

---

**Note:** If you are working behind a firewall, some additional configuration is required to access the Weather Web Service. Specifically, you will need to add HTTP proxy details to the `<proxyInfo>` tag in your Omniportlet producer's `provider.xml` file. This file is located at:

```
JDEV_
HOME\jdev\extensions\oracle.adfp.seededoc4j.10.1.3.2
.\j2ee\home\applications\portalTools\omniPortlet\WEB
-INF\providers\omniPortlet
```

When making changes to the `provider.xml` file, the portlet producer must be refreshed on the WebCenter application side. To update the proxy details and refresh the portlet producer, perform the following steps:

1. Stop the WebCenter application (consumer).
2. Update the `provider.xml` file and save your changes.
3. In Oracle JDeveloper, refresh the portlet producer on the WebCenter application (consumer) side.
4. Redeploy the WebCenter application.

For more information, see topic titled "B.2.1 Configuring the OmniPortlet Producer to Access Data Outside a Firewall" in *Oracle WebCenter Framework Developer's Guide*.

---

1. Display `MyWeather.jspx` in a browser. In the Applications Navigator, right-click **MyWeather.jspx**, and choose **Run**.
2. In the browser, click the **Define** link to start the OmniPortlet wizard.
3. Choose **Web Service**, and click **Next**.

After you finish this tutorial, you'll probably want to learn how to use other data sources with OmniPortlet, such as spreadsheets (character-separated values), XML, and even application data from existing Web pages. All of these options are described in the *Oracle WebCenter Framework Developer's Guide*.

4. Enter the **WSDL URL** of a demonstration weather Web Service available through Oracle Technology Network at:

```
http://webservices.oracle.com/WeatherWS/WeatherWS?WSDL
```

This Web Service has one method (`WeatherWS.giveMeSomeWeatherInfo`) and accepts one parameter (`param0`). When you pass a valid US ZIP Code through `param0`, the Web Service returns weather information for the area specified.

**Note:** This Web Service does not return live weather information. It is useful for demonstration purposes only.

5. To access this method parameter, click **Show Methods**.  
A parameter labeled `param0` should display on the page.
6. Enter `##Param1##` in the `param0` parameter field.

This tells the Web Service to accept values from the Omniportlet parameter `Param1`. Later on, you'll map portlet parameter `Param1` to a page variable named `ParameterFormPortlet1_1_ora_wsrp_navigparam_Parameter1`, see [Step 6: Configuring the Portlets Together](#).

7. Under Portlet Parameters, set a Default Value for Param1. Enter: 94065  
This tells the Web Service to return weather information for ZIP code 94065 (Redwood City, CA) by default.
8. Click **Next**, and then **Next** again as you don't need to set any filter options.
9. On the View Page, enter the values shown in [Table 6-1](#).

**Table 6-1 OmniPortlet View Page Settings**

Settings	Value
Title	Enter a title for the portlet: Weather Forecast
Header Text	Display the current value of Param1 (current ZIP code) in the portlet header. Enter: For ZIP Code ##Param1##

10. Click **Next**, then specify the Column Label, Column, and Display As properties for the weather data as shown in [Table 6-2](#).








**Table 6-2 Column Properties for the Weather Forecast Portlet**

Name	Column Label	Column	Display As
Field1	Day	dayOfWeek	Text
Field2	Hi	hiTemp	Text
Field3	Lo	lowTemp	Text
Field4	Sky	sky	Text
Field5	(Blank)	img	Image

11. Click **Finish** to display weather information for Redwood City. You should now see both portlets in your browser ([Figure 6-19](#)).



Figure 6–19 OmniPortlet Displaying a Weather Web Service

Weather Forecast			
For ZIP Code 94065			
Day	Hi	Low	Sky
Monday	55	45	Rainy 
Tuesday	85	80	Sunny 
Wednesday	61	50	Partly Cloudy 
Thursday	80	70	Cloudy 
Friday	63	50	Rainy 
Saturday	70	55	Rainy 
Sunday	68	50	Partly Sunny 

The Omniportlet displays a weather forecast for Redwood City, CA, which has the ZIP code 94065. The Web Service providing this data can return weather information for other areas if it is passed a valid US ZIP code. In the next step you'll learn how to link these two portlets and make the first portlet (Parameter Form Portlet) drive the content of the Omniportlet.

## Step 6: Configuring the Portlets Together

In this step you'll enable inter-portlet communication between two portlets.

1. In JDeveloper, display the page definition for MyWeather.jspx. In the Applications Navigator, right-click **MyWeather.jspx**, and select **Go to Page Definition**.
2. Examine MyWeatherPageDef.xml through the Structure window. In the executables section you'll find:
  - ParameterFormPortlet1\_1 - with three portlet parameters `ora_wsrp_navigparam_Parameter1`, `ora_wsrp_navigparam_Parameter2` and `ora_wsrp_navigparam_Parameter3`
  - OmniPortlet1\_1 - with five portlet parameters `Param1 - Param5`
  - variables - eight page variables, each mapped to a different portlet parameter. JDeveloper creates these mappings automatically when portlets (with parameters) are placed on a page

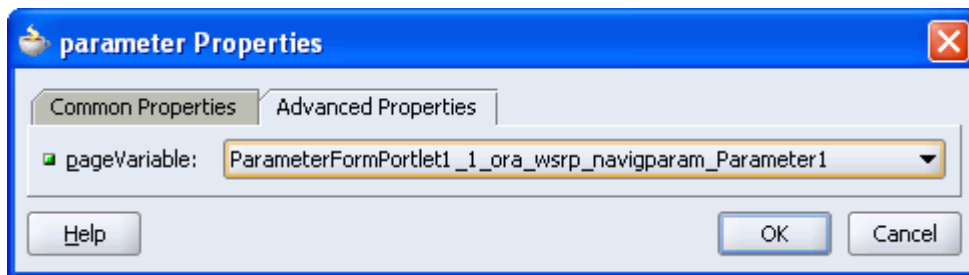
Although each portlet supports several parameters you'll only use *one* from each portlet to demonstrate inter-portlet communication:

- ora\_wsrp\_navigparam\_Parameter1 - you configured the Parameter Form Portlet to accept a ZIP Code through this parameter ([Step 3: Customizing the Parameter Form Portlet](#)).
- Param1 - you configured OmniPortlet to accept a ZIP code through this parameter ([Step 5: Building an OmniPortlet That Uses a Web Service](#)).

In the next step you'll map these two parameters to the same *page variable* and in doing so, facilitate parameter passing between two portlets. At the moment ora\_wsrp\_navigparam\_Parameter1 is mapped to page variable ParameterFormPortlet1\_1\_ora\_wsrp\_navigparam\_Parameter1, so let's map Param1 to this page variable too.

3. Use the Structure window to link OmniPortlet and the Parameter Form Portlet:
  - a. Right-click **Param1**, and select **Properties**.  
Param1 is located under MyWeatherPageDef\executables\OmniPortlet1\_1\parameters.
  - b. Click **Advanced Properties**.
  - c. From the pageVariable drop-down list, select **ParameterFormPortlet1\_1\_ora\_wsrp\_navigparam\_Parameter1** ([Figure 6–20](#)).

**Figure 6–20** *OmniPortlet Param1 Mapped to Same Page Variable as the Parameter Form Portlet*



- d. Click **OK**.
  4. Click **Save All** in the JDeveloper toolbar.
- Now, let's run the page to see the portlets working together.



## Step 7: Testing Portlet Interaction

Let's see the portlets on MyWeather.jspx working together.

1. Before running the page again, stop the embedded OC4J Server. Choose **Run, Terminate, Embedded OC4J Server** from the main menu.
2. In the Applications Navigator, right-click **MyWeather.jspx**, and select **Run**.  
Assuming there aren't any compiler errors, the page opens in a new browser window and you should see a weather forecast for the Redwood City area (ZIP Code 94065).
3. In the **ZIP Code** field, enter: 10001  
This is the ZIP Code for New York.
4. Click **OK**.

The Weather Forecast portlet should change, showing weather information for New York (something like [Figure 6-21](#)).

**Figure 6-21** Portlets Show Weather for New York

Weather Forecast			
For ZIP Code 10001			
Day	Hi	Low	Sky
Monday	55	45	Rainy 
Tuesday	68	50	Partly Sunny 
Wednesday	70	55	Partly Cloudy 
Thursday	63	50	Partly Sunny 
Friday	80	70	Cloudy 
Saturday	61	50	Partly Cloudy 
Sunday	85	80	Sunny 

Congratulations! You've completed this lesson and made two portlets communicate with each other.

In the next lesson, you will learn how to publish file system content in your WebCenter application.



---

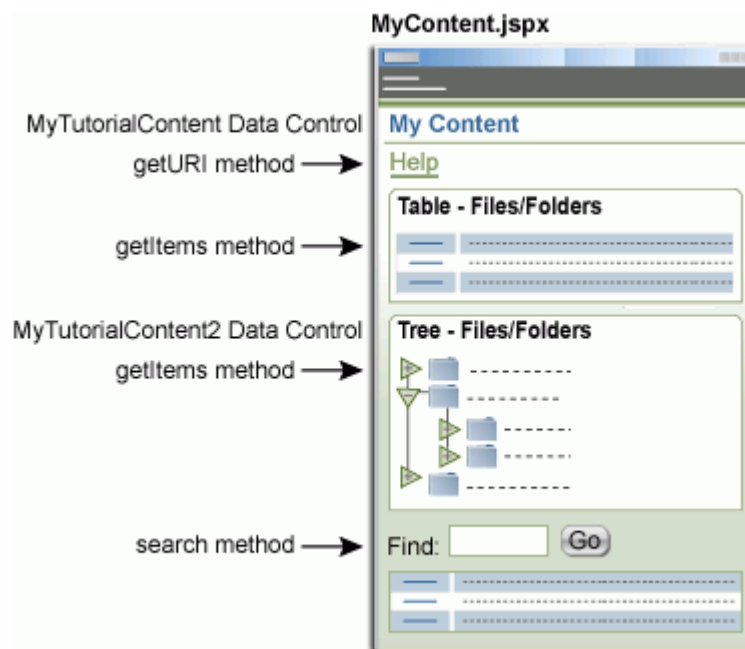
---

## Adding Content to Your Page

In this lesson you will learn how to publish content that resides on a file system in your WebCenter application. You'll see how easy it is to build content rich applications with Oracle JDeveloper and experiment with the various ways you can present file and folder information.

Figure 7-1 illustrates what your page (`MyContent.jspx`) will look like at the end of the lesson.

**Figure 7-1** *MyContent.jspx at the End of Lesson 7*



### Introduction

You'll add content to the tutorial application in the following steps:

- [Step 1: Creating a Data Control](#)
- [Step 2: Adding Content to a Page at Design Time](#)
- [Step 3: Displaying Folder Content in a Table](#)
- [Step 4: Displaying Folder Content in a Tree](#)
- [Step 5: Searching Folder Content](#)

## Prerequisites

During this lesson you'll need access to the sample you downloaded in [Chapter 2, "Getting Started"](#). Before you start, make a note of the location where the sample content is stored, see also [Downloading Sample Tutorial Files](#).

### Step 1: Creating a Data Control

In this step you'll define a data control that can access and publish some sample tutorial content stored on your file system. A *data control* is a container for all the data objects, collections, methods, and operations used to create UI components within your application.

1. In the Applications Navigator, right-click **Model**, and choose **New**.
2. Expand the **Business Tier** node and select **Content Repository**.
3. Select **Content Repository Data Control**, and then click **OK** to display the wizard.

This wizard creates a data control for a content repository, even when the content is on a file system.

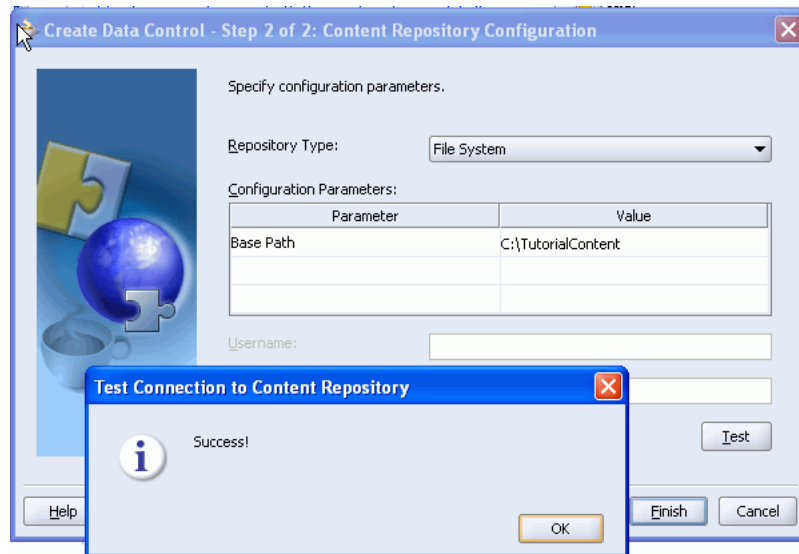
4. Click **Next** to skip the Welcome page.
5. Name the data control. Enter `MyTutorialContent`, and then click **Next**.
6. We want to publish content on your file system. Click the **Repository Type** drop-down list, and choose **File System**.

You can also set up a data control to access content in an Oracle Content DB repository, or an OracleAS Portal repository. For detailed information about how to set up these types of data controls, see the *Oracle WebCenter Framework Developer's Guide*.

7. In the Base Path field, enter the path to the sample content you unzipped earlier. For example: `C:\TutorialContent`

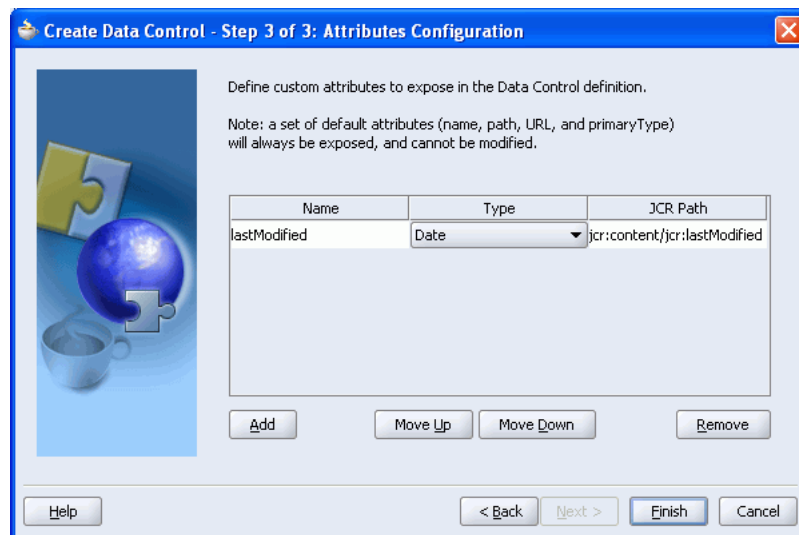
This shows the data control where to find your content.

8. Click the **Test** button to check whether you've entered the connection details correctly. You should see a `Success!` message like [Figure 7-2](#).

**Figure 7-2 File System Data Control - Testing the Connection**

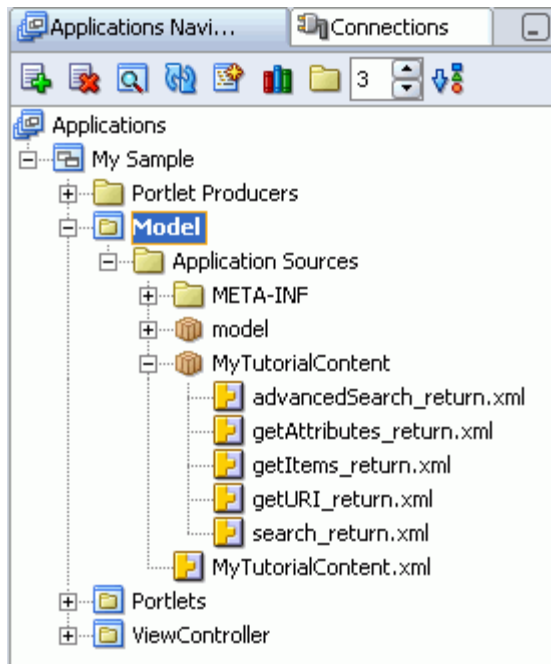
9. If you get an error message, click **OK** and edit the Base Path, taking care to specify the full path. If the test is successful, click **OK** to close the message box.
10. Click **Next**.

File system data controls expose several default attributes (`name`, `path`, `URI`, `primaryType`) and optionally, one custom attribute (`lastModified`)—see [Figure 7-3](#).

**Figure 7-3 File System Data Control - Custom Attribute Configuration**

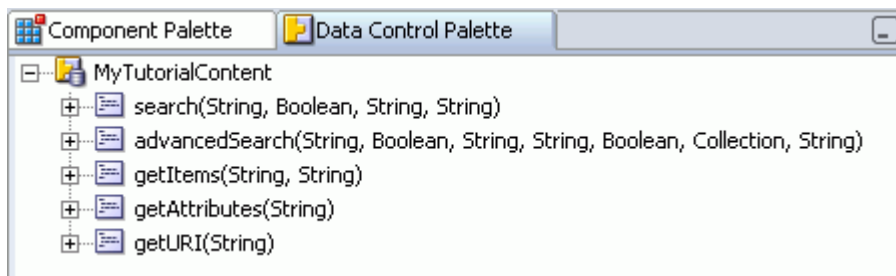
11. To accept the default attribute set, click **Finish**.

Take a look at the Applications Navigator. You should see several new entries under Model, Application Sources ([Figure 7-4](#)). For more information about data controls and the files they generate, see *Oracle WebCenter Framework Developer's Guide*.

**Figure 7-4 Files for File System Data Control MyTutorialContent**

12. The new data control should be available in the Data Control Palette, so let's display that now. From the **View** menu, select **Data Control Palette**.

Under *MyTutorialContent*, you should see a hierarchical list of objects, collections, methods, parameters, and operations for the new data control (Figure 7-5).

**Figure 7-5 Data Control Palette - MyTutorialContent**

File system data controls (such as *MyTutorialContent*) provide several methods for accessing and displaying file and folder information:

- **search** - Enables you to perform searches on the content exposed through the data control.
- **advancedSearch** - Enables you to perform advanced searches on the content exposed through the data control.
- **getItems** - Returns files and folders stored in a specific location of the content repository.
- **getAttributes** - Returns a list of attributes and their values for a given file or folder.
- **getURI** - Returns the URI of a file. In this release, direct access to *folders* through URIs is not supported.



You'll find detailed information about these methods in the *Oracle WebCenter Framework Developer's Guide*.

Let's see how to use some of these objects.

## Step 2: Adding Content to a Page at Design Time

In this step, you'll learn how to publish a hyperlink to a file using the data control method `getURI`. Let's start by creating a brand new page called `MyContent.jspx` on which you'll add a link to one of the sample files (`help.html`).

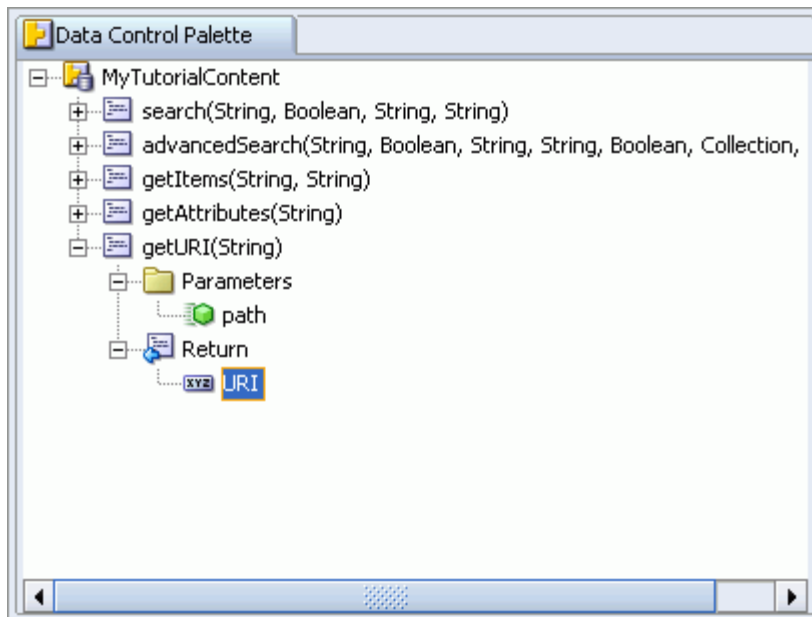
1. Create a new Java Server Faces page named `MyContent.jspx`.
  - a. In the Applications Navigator, right-click **ViewController**, and select **New**.
  - b. In the Categories pane, under Web Tier, select **JSF**.
  - c. Under Items, select **JSF JSP**, and click **OK**.
  - d. Click **Next** to skip the Welcome page.
  - e. In the File Name field, enter: `MyContent`
  - f. Under Type, click **JSP Document**, and then click **Next**.
  - g. Choose **Automatically Expose UI Components in a New Managed Bean**, and then click **Next**.
  - h. On the Tag Libraries page, make sure the following libraries appear in the **Selected Libraries** pane:
    - ADF Faces Components 10\_1\_3\_2\_0
    - ADF Faces HTML 10\_1\_3\_2\_0
    - ADF Portlet Components 10\_1\_3\_2\_0
    - Customizable Components Core 10\_1\_3\_2
    - JSF Core 1.0
    - JSF HTML 1.0
  - i. There is no need to set any other options, so click **Finish** on this page.

`MyContent.jspx` should open in the Visual Editor, ready for us to begin adding content from the file system. (If the page doesn't appear, click the **Design** tab.)

2. In the Data Control Palette, expand the **getURI(String)** node for `MyTutorialContent`.

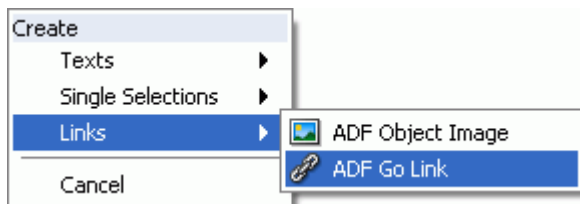
You should see one parameter (`path`), as well as one method return attribute `URI` listed here ([Figure 7-6](#)).

**Figure 7-6 Data Control Palette - MyTutorialContent.getURI**



3. Select the **URI** node, and drag and drop it onto MyContent.jspx.  
When you drag an item from the Data Control Palette and drop it on a page, JDeveloper displays a context menu of suitable components.
4. From the context menu, choose **Links**, and then **ADF Go Link**, as shown in [Figure 7-7](#).

**Figure 7-7 JDeveloper Context Menu for getURI**



5. To publish a link to a Help page located at C:\TutorialContent\help.html, use the **path** parameter to point to the file by entering: /help.html  
Make sure that you include a forward slash ([Figure 7-8](#)).

**Figure 7-8 GetURI - Action Binding Editor**



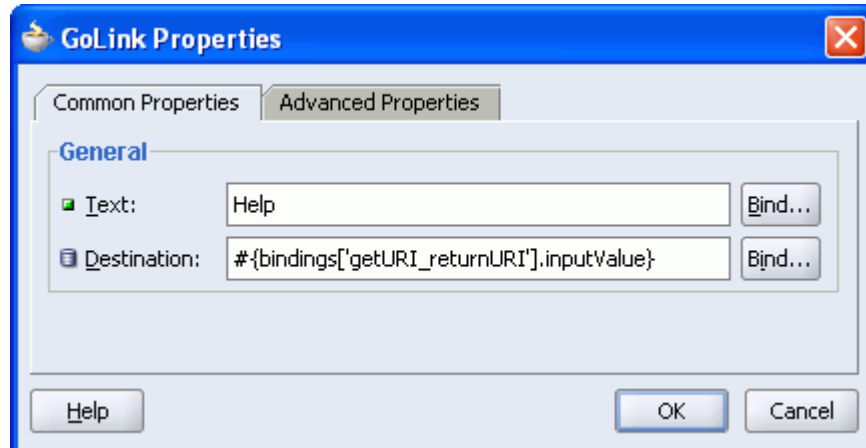
6. Click **OK**.

You should see a new goLink on MyContent.jspx with default link text goLink1.

7. In the Structure window, double-click **af:goLink - goLink1** to edit the default properties.
8. The default link text is goLink1. Replace this text with the word Help (Figure 7-9).

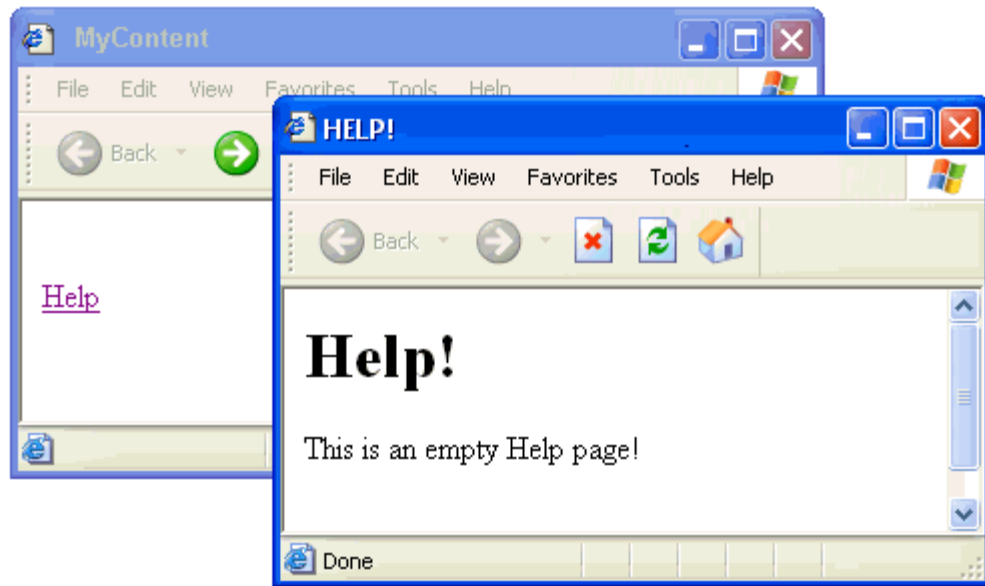
The expression in the Destination field, `#{bindings['getURI_returnURI'].inputValue}`, fetches the URI of the Help page.

**Figure 7-9 GoLink Properties - Configuring Help Link**



9. Click **OK** to close the GoLink Properties window.
10. Click the **Save All** icon in the JDeveloper toolbar.
11. Before running the page, stop the embedded OC4J Server. Choose **Run, Terminate, Embedded OC4J Server** from the main menu.
12. Right-click **MyContent.jspx**, and select **Run** to see the Help link in action.

When the page appears in your browser window, you should see the new Help link. Click the link to check that the correct file is displayed. Your browser should look like the one displayed in Figure 7-10.

**Figure 7–10 File Link Through the File System Data Control**

### Step 3: Displaying Folder Content in a Table

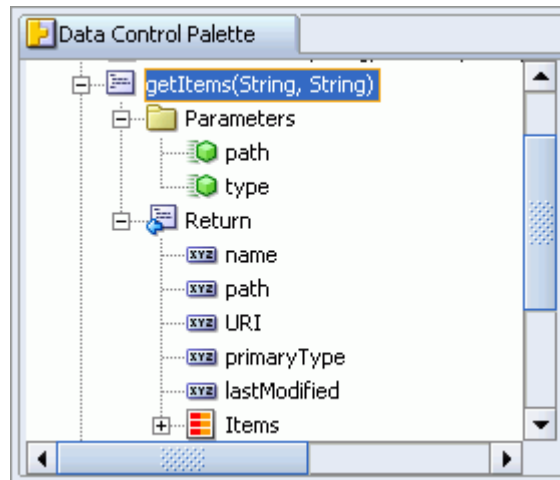
In this step, you'll learn how to publish file and folder information in a table using the data control method `getItems`. You'll then create a table (similar to the one shown in [Figure 7–11](#)) that lists all the files located at `C:\TutorialContent`. Each file name in the table will contain a hyperlink to the actual file.

**Figure 7–11 File System Content Displayed as Hypertext Links**

<b>My Tutorial Files</b>
<a href="#">help.html</a>
<a href="#">welcome.html</a>
<a href="#">hula.jpg</a>
<a href="#">camera.jpg</a>

1. In the Applications Navigator, double click **MyContent.jspx** to open the page in the Visual Editor.
2. In the Data Control Palette, expand the **getItems** node for **MyTutorialContent**. You should see two parameters (`path` and `type`) and the `Return` options shown in [Figure 7–12](#).

Figure 7-12 Data Control Palette - MyTutorialContent.getItems

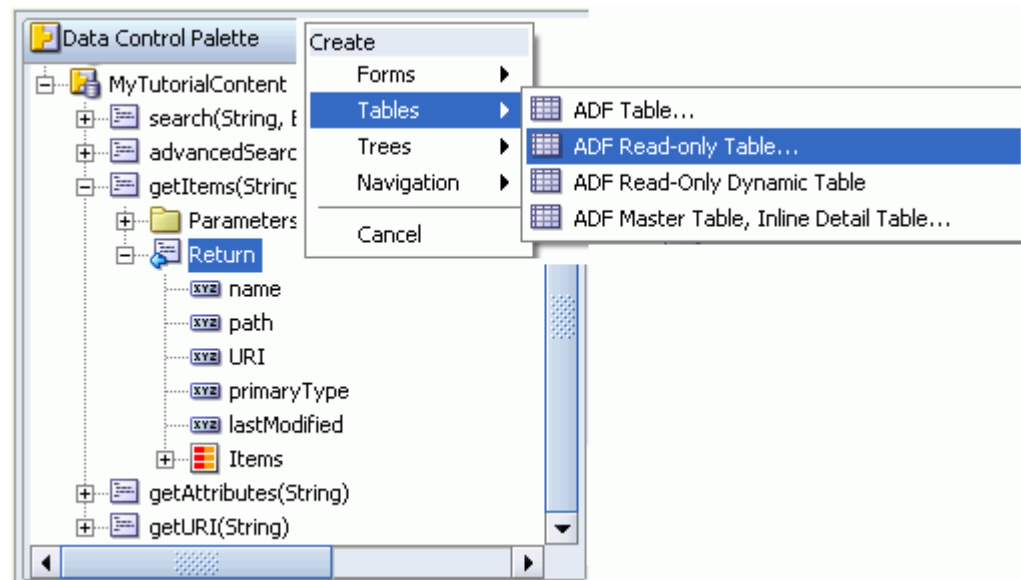


3. First, let's create a table that lists every file and folder available through this data control. To do this, highlight the **Return** node, drag it onto the page, and drop it below the Help link ([af:goLink - Help](#)).

When you drag an item from the Data Control Palette and drop it on a page, JDeveloper displays a context menu of suitable UI components. File system items can be displayed in a form, table, tree, or navigation item.

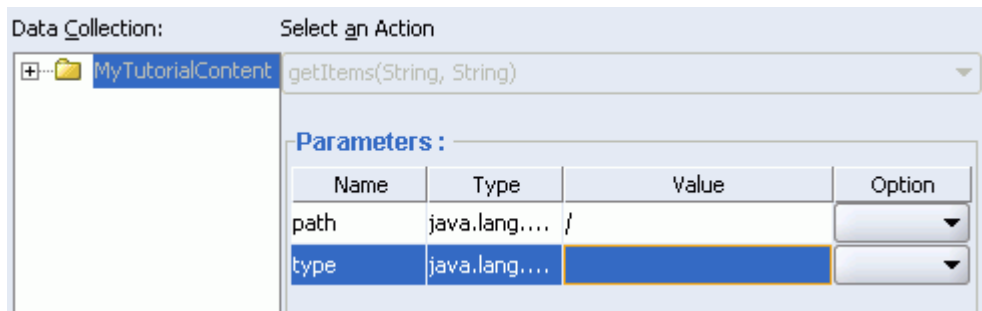
4. Choose **Tables** from the context menu, and then **ADF Read-only Table**, as shown in Figure 7-13.

Figure 7-13 JDeveloper Context Menu for getItems



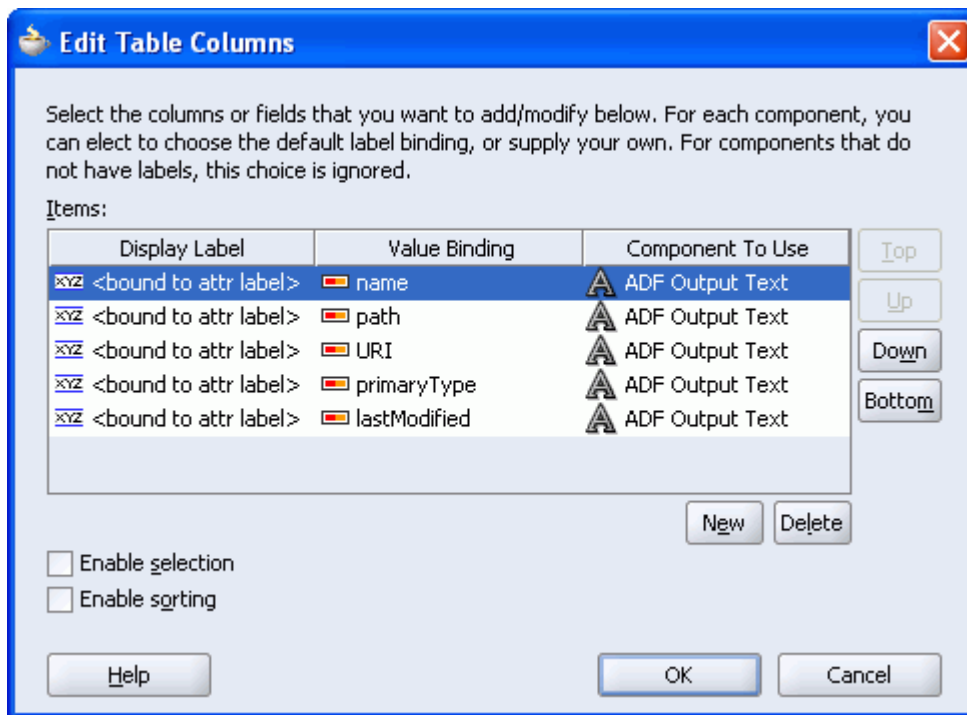
You should now see the Action Binding Editor (Figure 7-14).

5. To display everything under the data control's base path, enter `/` for the `path` parameter. Make sure that you enter a forward slash, not a backslash here.
6. Leave the `type` blank for now. This implies that the table must display both files and folders. Later on you'll configure the table to show files only.

**Figure 7–14** *getItems - Action Binding Editor*

7. Click **OK**.

You should see the Edit Table Columns window (Figure 7–15).

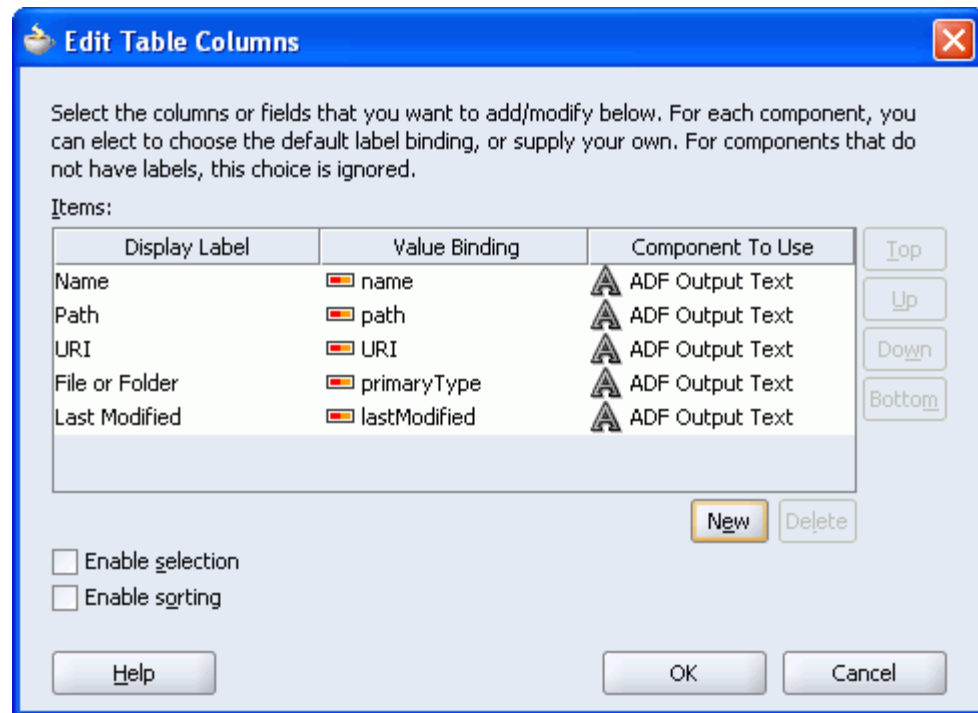
**Figure 7–15** *Edit Table Columns - Defaults*

Let's take a quick look at some of the display options available on this page. In addition to the file/folder name (`name`), you can publish additional content attributes such as `path`, `URI`, `primaryType`, and `lastModified`. All attributes are published by default, with default labels but you can customize the content displayed. You can remove attributes, edit display labels, and change the display order to suit your precise requirements.

For now, let's just edit the Display Labels to make them more meaningful.

8. Click **<bound to attr label>** (next to **name**) and enter: Name

Now edit the Display Label for the other attributes (`path`, `URI`, `primaryType`, `lastModified`). Enter new Display Labels such as `Path`, `URI`, `File` or `Folder`, and `Last Modified` as shown in Figure 7–16.

**Figure 7-16 Edit Table Columns - Customizing Display Labels**

9. Click **OK**.

You should see a table on `MyContent.jspx` that looks something like [Figure 7-17](#).

**Figure 7-17 Read-Only Table for Publishing Folder Content**

Name	Path	URI	File or Folder	Last Modified
<code>{row.name}</code>	<code>{row.path}</code>	<code>{row.URI}</code>	<code>{row.primaryType}</code>	<code>{row.lastModified}</code>
<code>{row.name}</code>	<code>{row.path}</code>	<code>{row.URI}</code>	<code>{row.primaryType}</code>	<code>{row.lastModified}</code>
<code>{row.name}</code>	<code>{row.path}</code>	<code>{row.URI}</code>	<code>{row.primaryType}</code>	<code>{row.lastModified}</code>

10. Let's see the page in a browser. Right-click `MyContent.jspx`, and select **Run**.

When the page appears in your browser window, you should see a list of all the file and folders available through the `MyTutorialContent` data control (for example, everything under the directory `C:\TutorialContent`) as shown in [Figure 7-18](#).

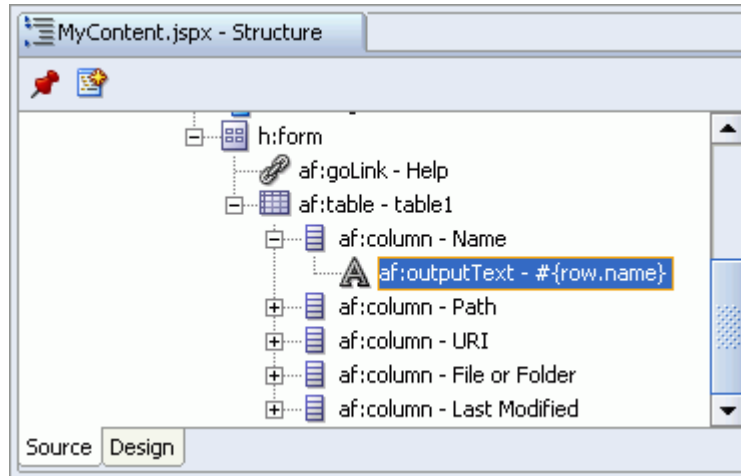
**Figure 7-18 Browser - Folder Content Displayed in a Table**

Name	Path	URI	File or Folder	Last Modified
<code>More_Content</code>	<code>/More_Content</code>	<code>/get/conn/MyTutorialContent/path/More_Content</code>	nt:folder	
<code>More_Images</code>	<code>/More_Images</code>	<code>/get/conn/MyTutorialContent/path/More_Images</code>	nt:folder	
<code>help.html</code>	<code>/help.html</code>	<code>/get/conn/MyTutorialContent/path/help.html</code>	nt:file	25/07/2006
<code>welcome.html</code>	<code>/welcome.html</code>	<code>/get/conn/MyTutorialContent/path/welcome.html</code>	nt:file	25/07/2006
<code>hula.jpg</code>	<code>/hula.jpg</code>	<code>/get/conn/MyTutorialContent/path/hula.jpg</code>	nt:file	26/07/2006
<code>camera.jpg</code>	<code>/camera.jpg</code>	<code>/get/conn/MyTutorialContent/path/camera.jpg</code>	nt:file	26/07/2006

By default, the table displays file/folder attributes as read-only text (`af:outputText`). In the next step, you'll learn how to display the Name attribute as an ADF GoLink (`af:goLink`).

11. In the Structure window as shown in [Figure 7-19](#), expand the first column of the table (`af:column - Name`) to reveal the default display format `af:outputText - #{row.name}`.

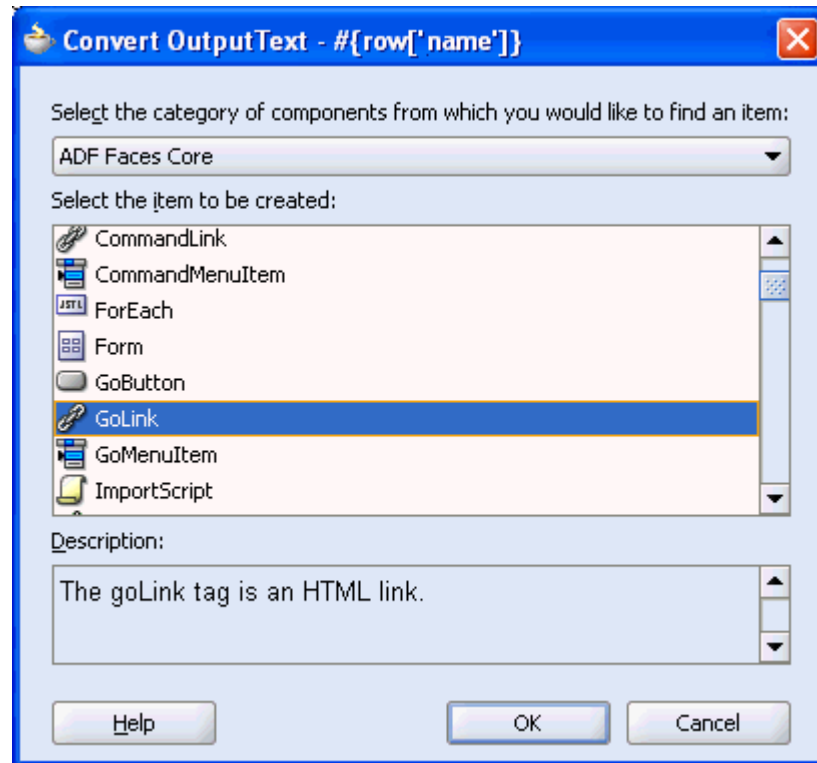
**Figure 7-19** Default Display Format for the Name Column



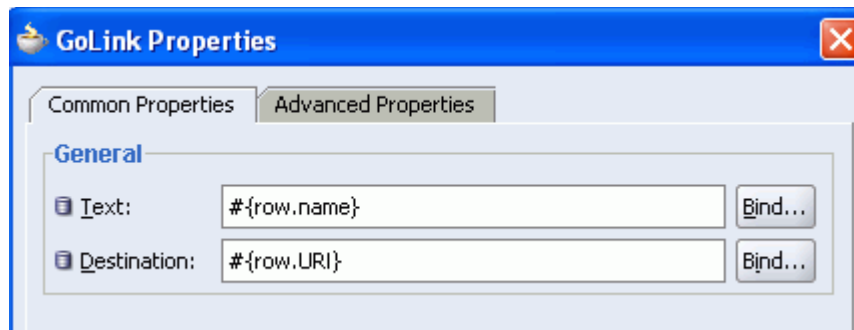
12. Right-click `af:outputText - #{row.name}`, and click **Convert**.
13. From the pull-down menu, choose **ADF Faces Core**.
14. Choose **GoLink** ([Figure 7-20](#)).



Figure 7-20 Convert OutputText to a GoLink



15. Click **OK**, and then click **OK** again to confirm the conversion.  
For this tutorial, we want the GoLinks to display file/folder names (name) as hyperlinks. To do this, you'll need to edit the default GoLink properties.
16. In the Structure window, right-click the new **af:goLink - goLink 1**, and click **Properties** to display the default settings.
17. The default link text is *goLink 1*. To display file/folder names instead (name), use the binding editor to build the required expression `#{row.name}` as shown in Figure 7-21:
  - a. For the **Text** field, click **Bind**.
  - b. Expand **JSP Objects**, and then expand **row**.
  - c. Double-click **name** to select the expression `#{row.name}`.
  - d. Click **OK**.
18. To generate URIs for the HTTP hyperlinks:
  - a. For the **Destination** field, click **Bind**.
  - b. Expand **JSP Objects**, and then expand **row**.
  - c. Double-click **URI** to select the expression `#{row.URI}`.
  - d. Click **OK**.

**Figure 7-21 GoLink Properties - Configuring File Names as Hyperlinks**

19. Before running the page again, stop the embedded OC4J Server. Choose **Run, Terminate, Embedded OC4J Server** from the main menu.
20. Run `MyContent.jspx` again.  
This time you should see a list of hyperlinked file/folder names like the one shown in [Figure 7-22](#).

**Figure 7-22 Browser - Folder Content Displayed as Hyperlinks**

Name	Path	URL	File or Folder	Last Modified
<a href="#">More_Content</a>	/More_Content	/get/conn/MyTutorialContent/path/More_Content	nt:folder	
<a href="#">More_Images</a>	/More_Images	/get/conn/MyTutorialContent/path/More_Images	nt:folder	
<a href="#">help.html</a>	/help.html	/get/conn/MyTutorialContent/path/help.html	nt:file	25/07/2006
<a href="#">welcome.html</a>	/welcome.html	/get/conn/MyTutorialContent/path/welcome.html	nt:file	25/07/2006
<a href="#">hula.jpg</a>	/hula.jpg	/get/conn/MyTutorialContent/path/hula.jpg	nt:file	26/07/2006
<a href="#">camera.jpg</a>	/camera.jpg	/get/conn/MyTutorialContent/path/camera.jpg	nt:file	26/07/2006

21. Click one of the *file* names.  
The file you pick should display in a browser window.
22. Click the name of a *folder*.  
This time you'll see an authorization error because you cannot access a folder through a direct URL. Folders may be accessed through a data control only.  
Before moving onto the next step, let's summarize what you've done so far. First you created a table based on the MyTutorialContent data control. You saw how, by default, the table publishes file system information in plain, unformatted text. You then applied some formatting to the file and folder names to display them as hypertext links.  
To finish off, let's tidy up the table. Let's configure the table to show only the Name column, and limit the display to files only.
23. Back in JDeveloper, configure the table to show the Name column only:
  - a. In the Structure window, right-click the `af:table -table1` node, and choose **Properties**.
  - b. Click the **Column Summary** tab. Use the **Delete** button to remove all but the Name column ([Figure 7-23](#)).

**Figure 7-23 Table Properties - Editing Columns**

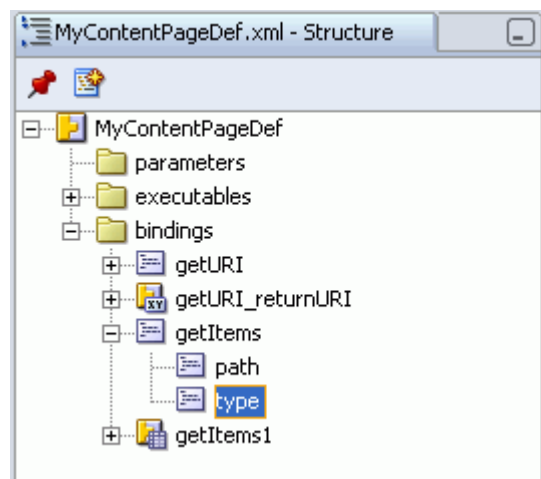
Header	Value	Component
Name	{row.name}	af:goLink

- c. Click the **Column Details** tab.
- d. For Header Text, enter My Tutorial Files as shown in [Figure 7-24](#).

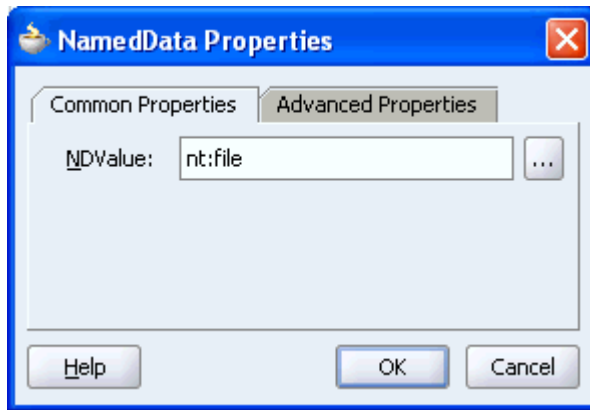
**Figure 7-24 Table Properties - Editing Column Display Options**

Columns:																													
My Tutorial Files	<table border="1"> <tr> <td>Top</td> <td>Header Text:</td> <td>My Tutorial Files</td> <td>Bind...</td> </tr> <tr> <td>Up</td> <td>Component Value:</td> <td>{row.name}</td> <td>Bind...</td> </tr> <tr> <td>Down</td> <td>UIComponent:</td> <td>af:goLink</td> <td></td> </tr> <tr> <td>Bottom</td> <td>Format Type:</td> <td></td> <td></td> </tr> <tr> <td></td> <td><input type="checkbox"/> Is Sortable</td> <td></td> <td></td> </tr> <tr> <td>Group</td> <td>Sort Property:</td> <td>name</td> <td></td> </tr> <tr> <td>Ungroup</td> <td></td> <td></td> <td></td> </tr> </table>	Top	Header Text:	My Tutorial Files	Bind...	Up	Component Value:	{row.name}	Bind...	Down	UIComponent:	af:goLink		Bottom	Format Type:				<input type="checkbox"/> Is Sortable			Group	Sort Property:	name		Ungroup			
Top	Header Text:	My Tutorial Files	Bind...																										
Up	Component Value:	{row.name}	Bind...																										
Down	UIComponent:	af:goLink																											
Bottom	Format Type:																												
	<input type="checkbox"/> Is Sortable																												
Group	Sort Property:	name																											
Ungroup																													
New	Delete																												

- e. Click **OK**.
24. To configure the table to show files only (not folders), you'll need to edit the page definition file:
- a. Right-click **MyContent.jspx**, and select **Go to Page Definition**.
  - b. In the Structure window, expand **bindings** and **getItems** ([Figure 7-25](#)).

**Figure 7-25 Configure the type Property in MyContentPageDef.xml**

- c. Double-click **type**.
- d. The type options are `nt:file` and `nt:folder`. To specify display files only, enter `nt:file` in the **NDValue** field, and click **OK** as shown in [Figure 7-26](#).

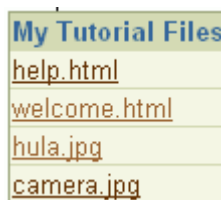
**Figure 7–26 NDValue - Configuring File Display Only**

25. Let's take a moment to examine a couple of binding settings for the data control that are set by default in the page definition file. You may want to use these settings for your own application, so it's important to understand what they do:
  - a. In Structure view, expand the **executables** section.
  - b. Select the method iterator **getItemsIter**.
  - c. From the **View** menu, choose **Property Inspector** to view the default settings.
  - d. The **RangeSize** property controls how many file/folder items are displayed on each page. For this tutorial, let's keep the default RangeSize 10.
  - e. The **CacheResults** property determines whether table content is cached. By default, the results are cached (`true`) but caching may not be desirable for some applications where content changes frequently and real time updates are critical. Set this property to `false`.

When you run the page, you'll see that the file list updates dynamically whenever the page is refreshed.

26. Click the **Save All** icon in the JDeveloper toolbar.
27. Run the page again.

The list of files should look something like [Figure 7–27](#). This time, no folders are displayed.

**Figure 7–27 Browser - Files Only Displayed as Hypertext Links**

28. Test that caching is disabled. Rename one of the files on your file system, and then refresh the browser.

As you chose not to cache the table content, you'll see the new file name immediately.

In this lesson you learnt how to publish file system content in a table. In the next lesson you'll display the same content in a tree.

## Step 4: Displaying Folder Content in a Tree

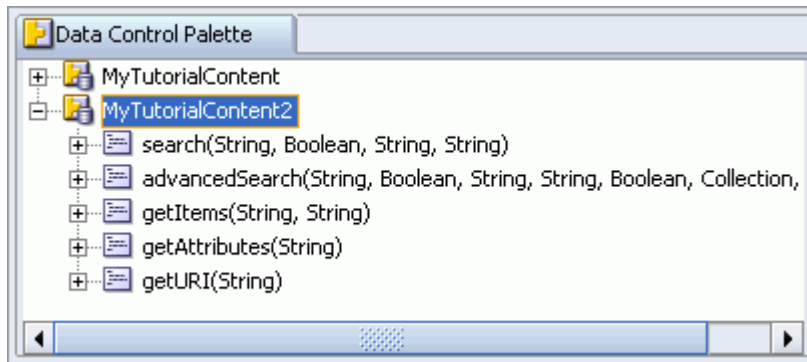
In this step you'll publish file and folder content in a hierarchical tree format using the `getItems` method. You'll create a tree (similar to the one shown in [Figure 7-28](#)) displaying files located at `C:\TutorialContent` and each file name in the tree will provide a hyperlink to the actual file. But first, let's create a new data control.

**Figure 7-28** File System Content Published in a Tree



1. In the Applications Navigator, right-click **Model**, and choose **New**.
2. Expand the **Business Tier** node, and select **Content Repository**.
3. Select **Content Repository Data Control**, and then click **OK**.
4. Click **Next** to skip the Welcome page.
5. Name the data control. Enter `MyTutorialContent2`, and then click **Next**.
6. We want to use this data control to publish content from a file system, so click the **Repository Type** drop-down list, and choose **File System**.
7. In the Base Path field, enter the path to the content you unzipped earlier, for example `C:\TutorialContent`, and then press **Enter**.
8. Click the **Test** button to check whether you've entered the connection details correctly. You should see a `Success!` message.
9. If you get an error message, click **OK** and edit the Base Path, taking care to specify the full path. If the test is successful, click **OK** to close the message box, then click **Finish**.
10. The new data control should be available in the Data Control Palette, so let's display that now. From the View menu, select **Data Control Palette** ([Figure 7-29](#)).

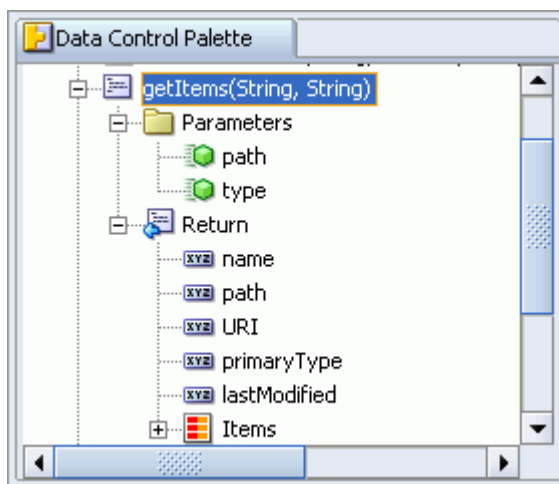
**Figure 7-29 Data Control Palette - MyTutorialContent2**



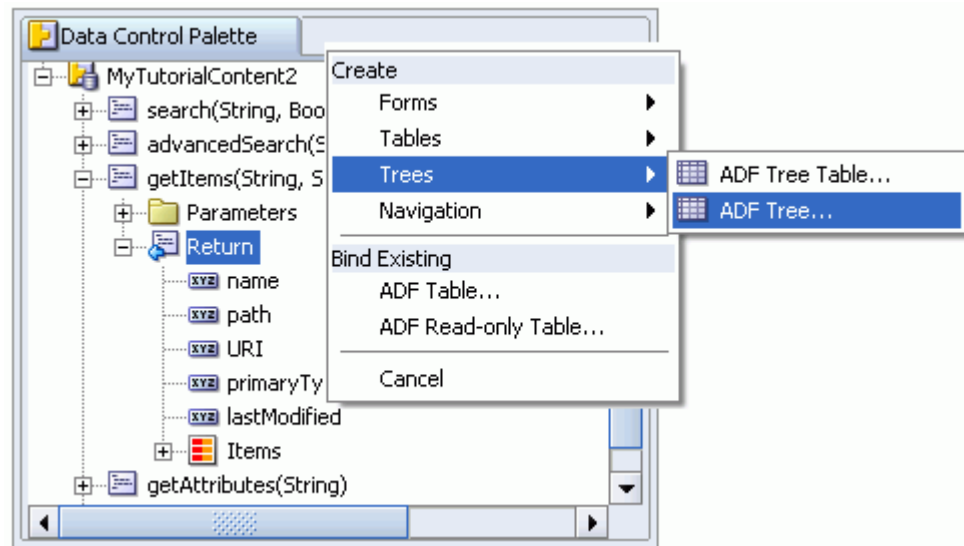
Now, let's publish content available through this data control in an hierarchal tree format.

11. In the Applications Navigator, double click **MyContent.jspx** to open the page in the Visual Editor.
12. In the Data Control Palette, expand the **GetItems** node under **MyTutorialContent2** (Figure 7-30).

**Figure 7-30 MyTutorialContent2.getItems**



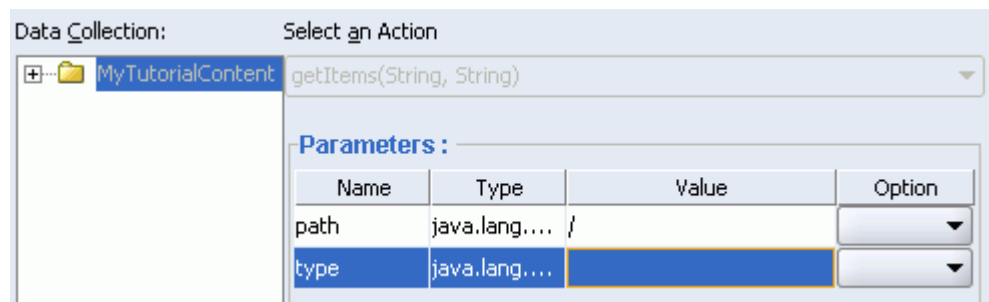
13. Highlight the **Return** node, drag it onto the page, and drop it below the table (**af:Table - table1**).
14. From the context menu, select **Create Trees, ADF Tree** as shown in Figure 7-31.

**Figure 7–31** JDeveloper Context Menu for `getItems`

You should see the Action Binding Editor.

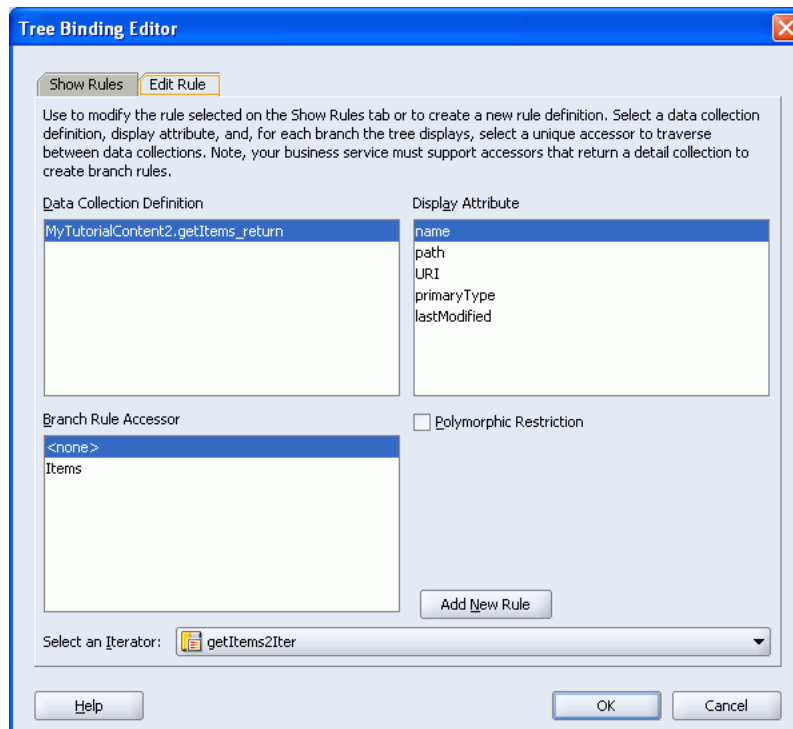
15. To create a tree that displays everything under the base path (C:\TutorialContent), enter / for the path parameter as shown in [Figure 7–32](#). Make sure you enter a forward slash, not a backslash.

Leave the type blank, as the tree must show both files and folders.

**Figure 7–32** Action Binding Editor

16. Click OK.

The Tree Binding Editor is displayed ([Figure 7–33](#)). Let's take a quick look at this page.

**Figure 7–33 Tree Binding Editor - Default Edit Rule Tab**

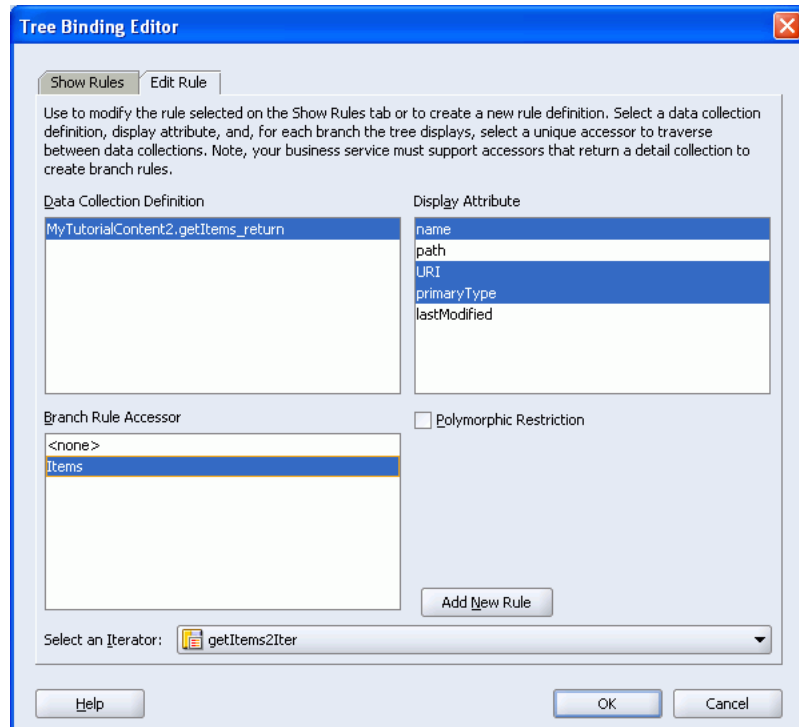
Rules define how tree data controls fetch and display hierarchical data. The default rule settings (Figure 7–33) show `getItems` is the root of the tree, that tree nodes (or branches) may display only the name attribute, and that tree nodes may not display children—Branch Rule Accessor is `<none>`.

In the next few steps you'll edit these default settings and define a rule that will expose files/folders under each node in a hierarchical format and enable you to hyperlink file names.

17. In the **Display Attribute** list, multi-select the attributes `name`, `URI`, and `primaryType` as shown in Figure 7–34.

You'll use the `name` attribute to display file/folder names in the tree, and you'll use the `URI`, and `primaryType` attributes to build hyperlinks to file content. The other attributes (`path` and `lastModified`) are not used in this tutorial so there is no need to select them here.



**Figure 7–34 Tree Binding Editor - Rule Selections**

18. For the Branch Rule Accessor, select **Items**.

This enables tree nodes to display any children that may exist in a hierarchical format.

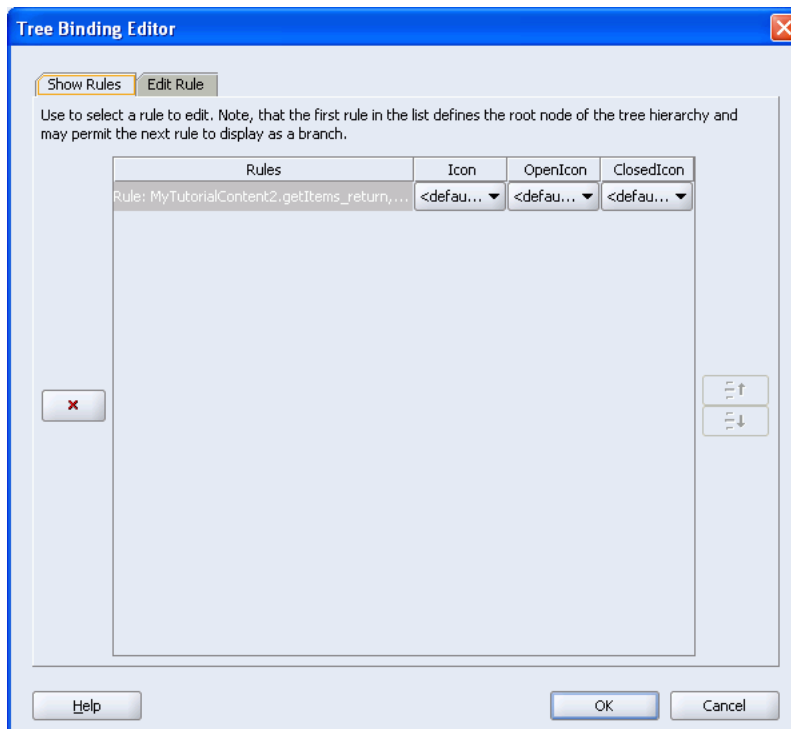
19. Click **Add New Rule**.

You should see the message shown in [Figure 7–35](#).

**Figure 7–35 New Rule Confirmation Dialog**

20. Click **OK** to close the message box and display the Show Rules tab ([Figure 7–36](#)).

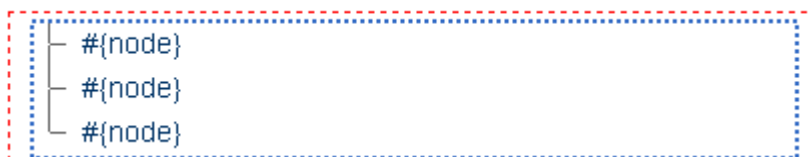
**Figure 7–36 Tree Binding Editor - Show Rules Tab**



21. Click **OK**.

Now you should see a tree structure similar to [Figure 7–37](#) displayed on `MyContent.jspx`.

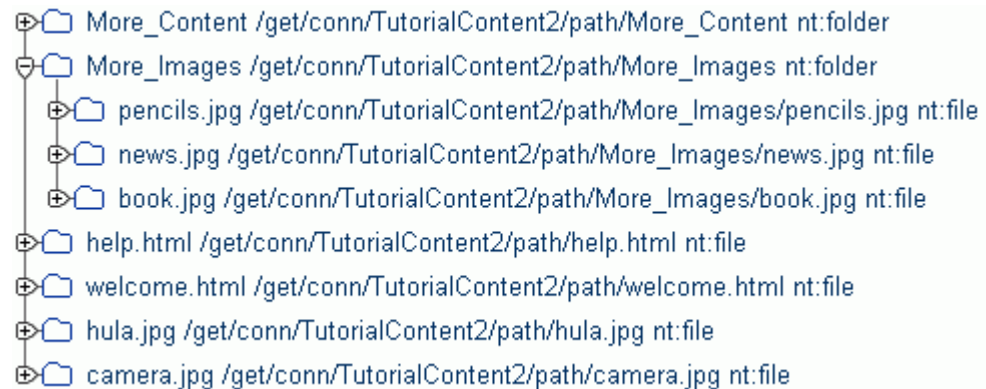
**Figure 7–37 Tree for Navigating Folder Content**



22. Before running the page again, stop the embedded OC4J Server. Choose **Run, Terminate, Embedded OC4J Server** from the main menu.

23. Let's see the page in a browser. Right-click `MyContent.jspx`, and select **Run**.

When the page appears in your browser window, you should see a list of files and folders available through the `MyTutorialContent2` data control, for example, everything under the directory `C:\TutorialContent`. Expand the `More_Images` node to see access content in this subdirectory ([Figure 7–38](#)).

**Figure 7–38 Folder Content Displayed in a Tree**

Now let's hide the URI and `primaryType` attribute. In a moment, you'll use these attributes to build hyperlinks but they don't need to be *displayed* in the tree.

24. In the Applications Navigator, select **MyContent.jspx**.
25. In Structure window, right-click **af: tree - tree1**, and choose **Properties**.
26. Change the **Node Stamp Value** to `#{node.name}`, and click **OK**.
27. Let's see what the tree looks like now. Right-click **MyContent.jspx**, and select **Run**.  
You should see file and folder names only (Figure 7–39).

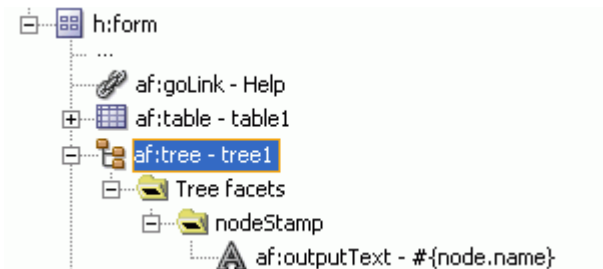
**Figure 7–39 Tree Displays File and Folder Names**

Trees will display the first ten items (by default) but you can customize this through the **RangeSize** property for the `GetItems` method in the page's definition file.

By default, the tree displays file/folder names as read-only text, but let's display them as hyperlinks as you did for the table. Like last time, you want to add hyperlinks to the file names (not folders). Unlike last time, you cannot hide the folders as they're required for navigation through the tree. As an alternative, you can display the folder names as read only text. To accommodate this dual functionality, let's use an ADF Faces Switcher component (`af:switcher`) with two facets - one for folders and one for files.

28. In the Structure window, navigate down to the **nodeStamp** facet to reveal the current display format `af:outputText - #{node.name}` (Figure 7–40).

**Figure 7–40 Default Display Format for Trees**



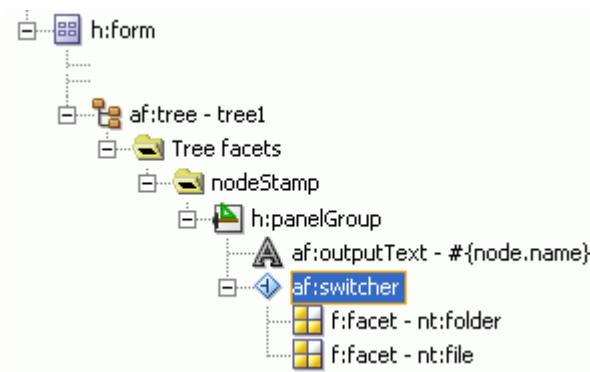
29. Add an ADF Switcher component:
  - a. Right-click **nodeStamp**, and choose **Insert Inside nodeStamp, ADF Faces Core**.
  - b. Choose **Switcher**, and then click **OK** (Figure 7–41).

**Figure 7–41 af:switcher Component**



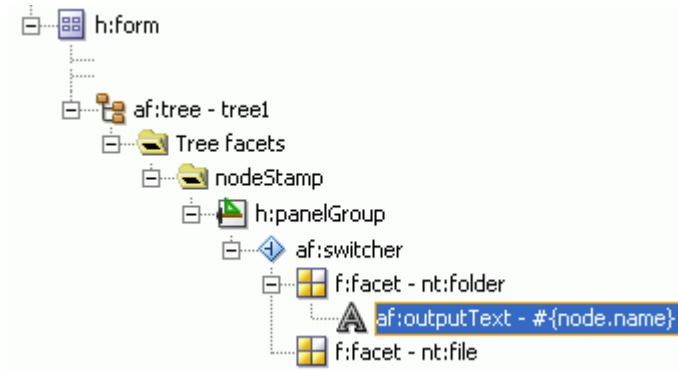
- c. Right-click **af:switcher**, and click **Properties**.
  - d. For **FacetName**, enter the expression: `#{node.primaryType}`
  - e. Click the **Advanced Properties** tab.
  - f. For **DefaultFacet**, enter `nt:file`, and then click **OK**.
30. Now insert two facets for the switcher:
  - a. Right-click **af:switcher**, choose **Insert Inside af:switcher, JSF Core, Facet**.
  - b. Name the first facet `nt:folder`, and click **OK**.
  - c. Now repeat these steps, adding a second facet named `nt:file` (Figure 7–42).

**Figure 7–42 Switcher Component with Two Facets**



31. Folder names require no additional formatting. Let's reuse the default display format `af:outputText - #{node.name}` to display folder names in plain text:
  - a. In the Structure Window, select **af:outputText - #{node.name}**.
  - b. Drag and drop the `af:outputText` component on top of **f:facet - nt:folder** (Figure 7-43).

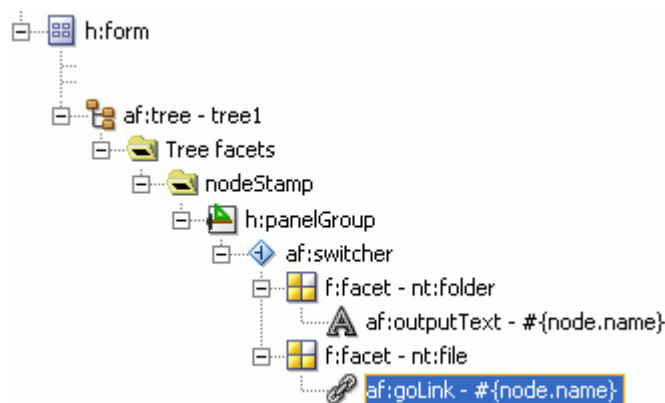
**Figure 7-43** Display `nt:folder` Facet as `outputText`



32. Use an `af:goLink` component to display file names as hyperlinks:
  - a. Right-click **f:facet - nt:file**, and then choose **Insert Inside f:facet - nt:file, ADF Faces Core**.
  - b. Choose **GoLink**, and click **OK**.
  - c. Right-click **af:goLink - goLink 1**, and click **Properties**.
  - d. For the link's Text, enter the expression: `#{node.name}`
  - e. For the link's Destination, enter the expression: `#{node.URI}`
  - f. Click **OK**.

Use the Structure window to check the `af:switcher` configuration. It should look like Figure 7-44.

**Figure 7-44** Switcher Showing Configured Facets



33. Click the **Save All** icon in the JDeveloper toolbar.
34. Run the page again.

This time, you'll see a tree of hyperlinked file names similar to [Figure 7–45](#).

**Figure 7–45 Content Published in a Tree**



35. Click any file name. You should see the content of that file displayed in a new browser window.

Notice that folder names are plain text, a result of our Switcher component.

## Step 5: Searching Folder Content

File system data controls provide a `Search` method for locating data and documents that are exposed through a data control. In this step you'll learn how to use this method to build a search form that enables users to search for content by file name and keyword, and display search results in a table.

At the end of this exercise, your search form will look similar to that shown in [Figure 7–46](#). Let's get started!

**Figure 7–46 Search Form Based On Data Control MyTutorialContent2**

Find files with all or part of this file name:

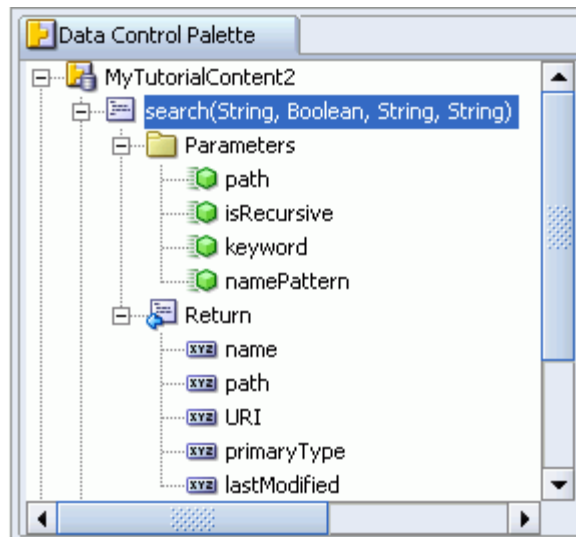
Use % for wildcard searches

Find files containing this word or phrase:

Files Found	Location	Last Modified
help.html	/help.html	25/07/2006
welcome.html	/welcome.html	25/07/2006
hula.jpg	/hula.jpg	26/07/2006
camera.jpg	/camera.jpg	26/07/2006
mycontent.html	/More_Content/mycontent.html	25/07/2006
pencils.jpg	/More_Images/pencils.jpg	27/06/2006
news.jpg	/More_Images/news.jpg	27/06/2006
book.jpg	/More_Images/book.jpg	27/06/2006

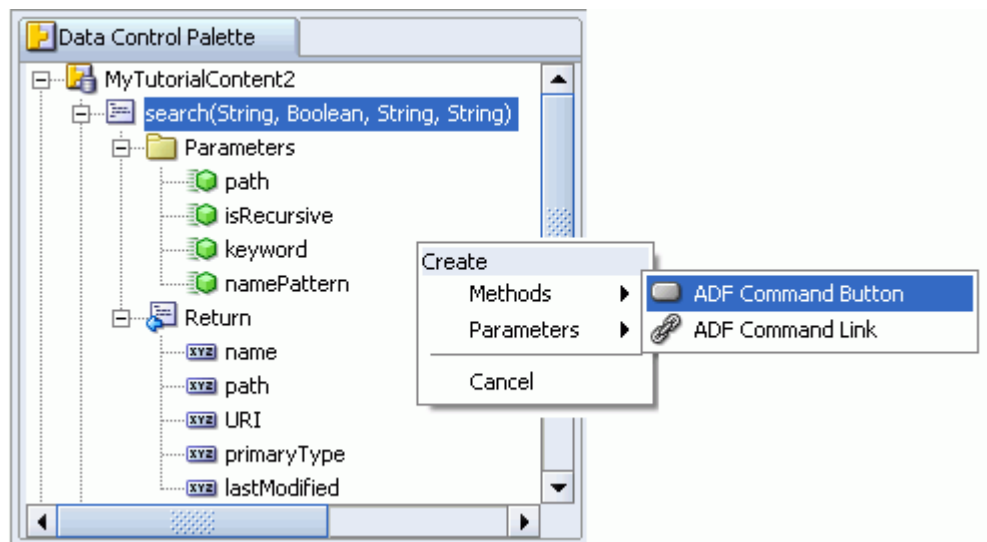
1. In the Data Control Palette, expand the **Search** node under **MyTutorialContent2** ([Figure 7–47](#)).

Figure 7-47 MyTutorialContent2.Search



2. Highlight the **search** node, drag it over to the Structure window, and drop it below the tree (af:Tree - tree1).
3. From the context menu, select **Create Method, ADF Command Button** as shown in Figure 7-48.

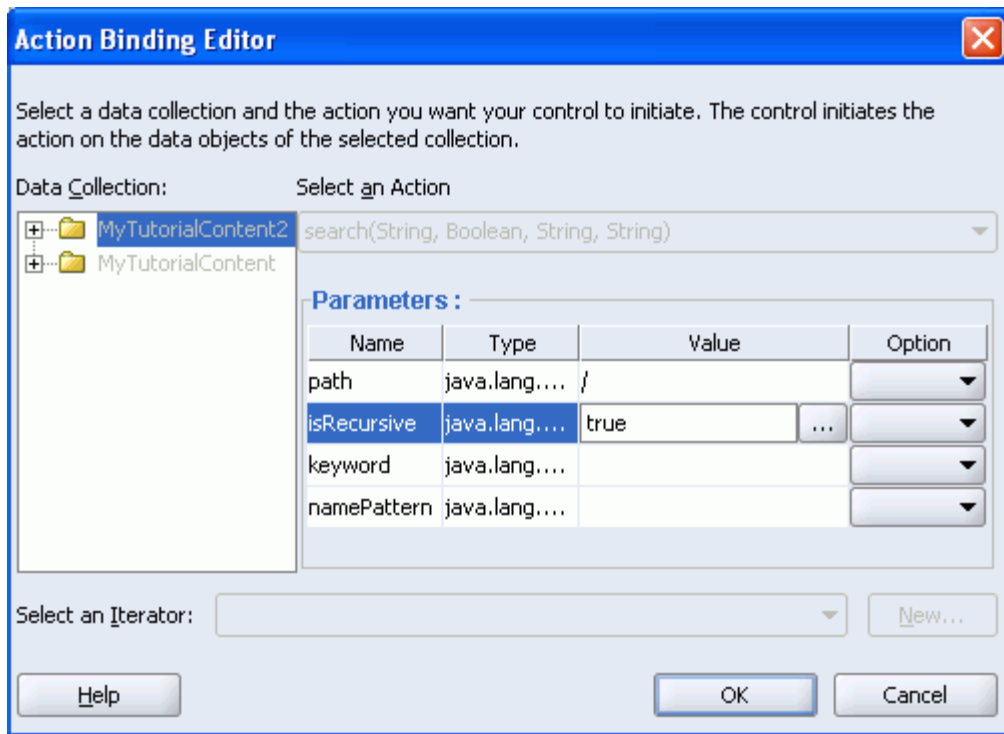
Figure 7-48 JDeveloper Context Menu for the Search Method



Now you should see the Action Binding Editor.

4. We want to search for everything under the base path (for example, C:\TutorialContent), so enter / for the path parameter. Make sure you enter a forward slash, not a backslash (Figure 7-49).

Figure 7-49 Action Binding Editor for Search Method



5. To automatically extend the search to *all* subdirectories under the location specified through the path attribute, enter `true` for the `isRecursive` parameter (Figure 7-49).

Leave the `keyword` and `namePattern` fields blank—you're not going to pre-define any search terms or filename criteria for this search. Instead, you'll provide input fields for both these parameters on the search form so that users can drive the search.

6. Click **OK**.

You should see a command button labeled `search` displayed on your page. Now let's add some input fields above this button into which a users may enter search criteria.

7. To enable users to search by file name, add the `namePattern` parameter:
  - a. In the Data Control Palette, expand the **Parameters** node, and then select **namePattern**.
  - b. Drag and drop the **namePattern** parameter *above* the Search button.
  - c. From the Create menu, choose **Texts, ADF Input Text w/ Label**.
  - d. In the Structure window, double-click **af:inputText #{bindings.namePattern...**, and enter a suitable label, for example: Find files with all or part of this file name:
  - e. For a Tip, enter: Use % for wildcard searches
  - f. Click **OK**.
8. To enable users to search for phrases or keyword within file content, add the `keyword` parameter to the page:



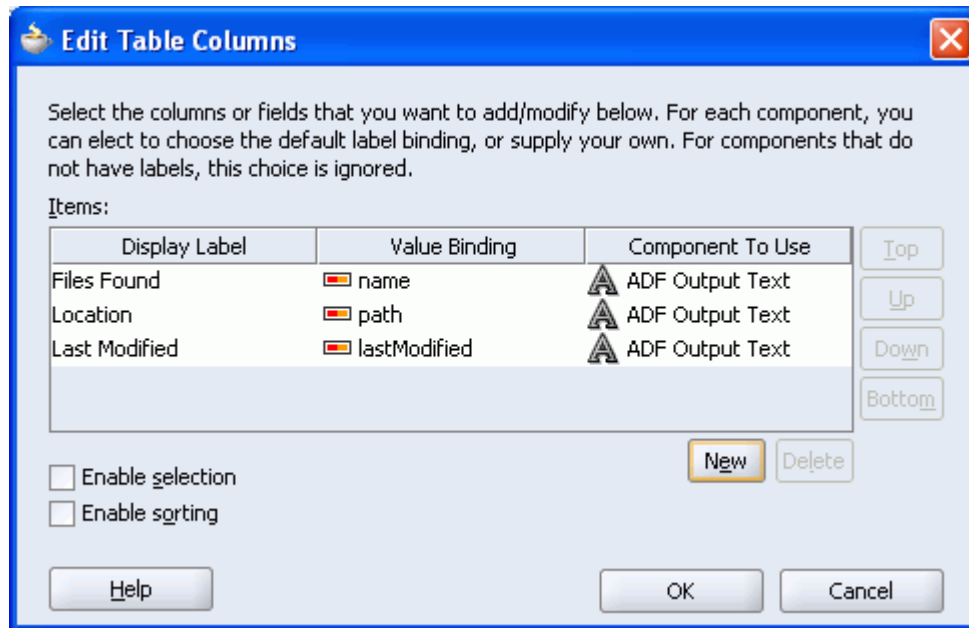
- a. In the Data Control Palette, expand the **Parameters** node, and select **keyword**.
- b. Drag and drop the **keyword** parameter above the Search button.
- c. From the Create menu, choose **Texts, ADF Input Text w/ Label**.
- d. In the Structure window, double-click **af:inputText #{bindings.keyword...**, and enter a suitable label such as:  
Find files containing this word or phrase:
- e. Click **OK**.

Your page should look like [Figure 7–50](#).

**Figure 7–50 Search Form With Two Input Parameters**

The screenshot shows a search form with two input fields and a search button. The first input field is labeled "Find files with all or part of this file name:" and contains the binding expression `#{bindings.namePattern.inputValue}`. Below this field is a small blue link that says "Use % for wildcard searches". The second input field is labeled "Find files containing this word or phrase:" and contains the binding expression `#{bindings.keyword.inputValue}`. Below the second field is a "search" button.

9. Now let's add a table for the search results:
  - a. In the Data Control Palette, select the **return** node below the Search method.
  - b. Drag the method **return**, and drop it below the Search button.
  - c. From the Create menu, choose **Tables, ADF Read-only Table**.
  - d. Use the Edit Table Columns dialog to edit some default display settings. First, delete the **URI** and **primaryType** columns.
  - e. Next, edit the Display Labels for the name, path, and lastModified columns. For example, enter the labels `Files Found`, `Location`, and `Last Modified` as shown in [Figure 7–51](#).

**Figure 7-51 Edit Display Labels for the Search Results Table**

- f. Click **OK**.
10. Click the **Save All** icon in the JDeveloper toolbar.  
Now let's display the search form in a browser and run some searches.
11. Before running the page again, stop the embedded OC4J Server. Choose **Run, Terminate, Embedded OC4J Server** from the main menu.
12. Right-click **MyContent.jspx**, and select **Run**.

You should see a search form similar to that shown in [Figure 7-52](#).

**Figure 7-52 Search Form Based On Data Control MyTutorialContent2**

Find files with all or part of this file name:

[Use % for wildcard searches](#)

Find files containing this word or phrase:

Files Found	Location	Last Modified
help.html	/help.html	25/07/2006
welcome.html	/welcome.html	25/07/2006
hula.jpg	/hula.jpg	26/07/2006
camera.jpg	/camera.jpg	26/07/2006
mycontent.html	/More_Content/mycontent.html	25/07/2006
pencils.jpg	/More_Images/pencils.jpg	27/06/2006
news.jpg	/More_Images/news.jpg	27/06/2006
book.jpg	/More_Images/book.jpg	27/06/2006

13. The first field on the form exposes the `namePattern` parameter. Enter some file name criteria, click the search button, and examine the content displayed in the

search result table. Only files matching the name pattern you specify should be displayed.

Try the following searches:

- To search for files with the .html file extension, enter `%html`. Three files should match this pattern.
  - To search for filenames containing the word *help*, enter `%help%`. Only `help.html` should match.
- 14.** The second field on the form exposes the `keyword` parameter. This parameter enables you to search for content inside files. Enter a word or phrase, click the search button, and examine the content displayed in the search result table. Only files containing the words you specify should be displayed.

Remove `%help%` from the field above, and try the following keyword searches:

- To search for files containing the word *WebCenter*, enter `WebCenter`. Two files should display.
- To search for files containing the words *WebCenter* and *tutorial*, enter `WebCenter AND tutorial`. Only one file contains both these words.

This completes the content integration lesson. You've learnt how to publish file system content in a WebCenter application and discovered how to add search capabilities too. For more detailed information, on this subject, refer to the *Oracle WebCenter Framework Developer's Guide*.

In the next lesson you'll use Oracle ADF security to secure the pages you've created during this tutorial.

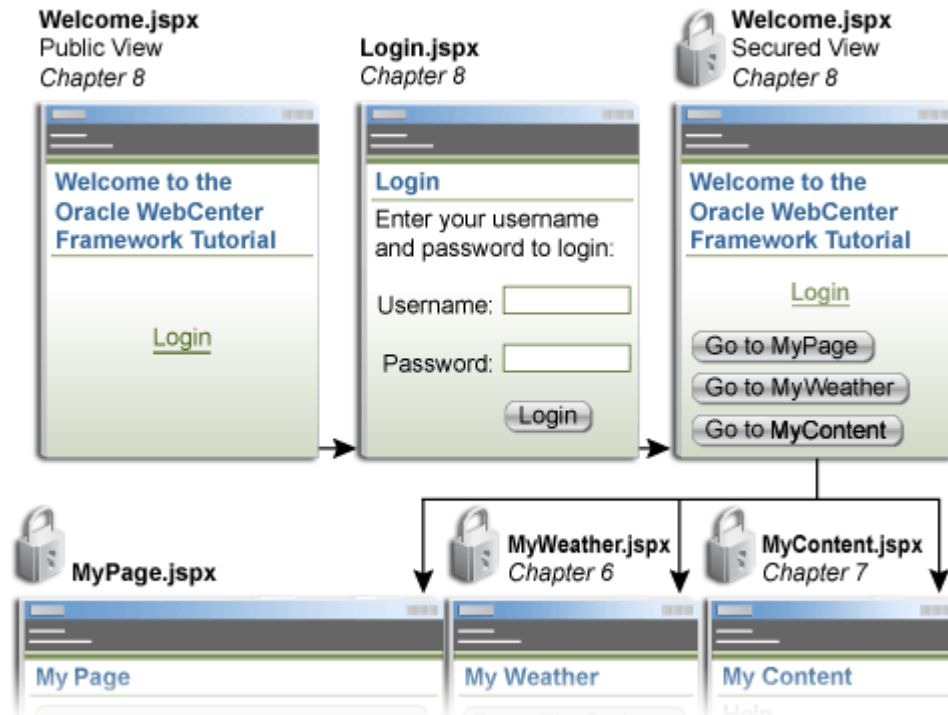


---

## Providing Security

In this lesson you'll learn how to secure the pages of a WebCenter application using Oracle ADF security. [Figure 8–1](#) shows you what the tutorial application will look like at the end of the lesson.

**Figure 8–1** Your Pages at the End of Lesson 8



### Introduction

We will add security to the tutorial application and test it in the following steps:

- [Step 1: Creating a Login Page](#)
- [Step 2: Configuring ADF Security Settings](#)
- [Step 3: Creating a Welcome Page](#)
- [Step 4: Securing Pages](#)
- [Step 5: Mapping Security Roles in orion-web.xml](#)

- [Step 6: Demonstrating the Security Features](#)
- [Step 7: Authorizing Access to Your Data Controls](#)

## Prerequisites

In this chapter you'll authenticate users against the lightweight XML resource provider `system-jazn-data.xml` supplied with the embedded OC4J. Before starting these exercises, you'll need to add the following user data to this file:

Role Name	Users	Description
page-viewer	Singh	This user may view secured pages.
page-personalizer	Cho	This user may personalize portlets on a secured page.
page-customizer	Harvey	This user may customize secured pages.
restricted-user	King	This user may not view secured pages.
users	Singh, Cho, King, Harvey, JtaAdmin, oc4jadmin	The users role maintains a list of every valid user.

Do *one* of the following:

- Copy the sample `system-jazn-data.xml` to the JDeveloper locations described in [Chapter 2 Copying the Sample system-jazn-data.xml File](#). The sample contains all the user data required to complete this chapter.
- Add the user data from scratch, following instructions in [Appendix A, "How to Set Up the Tutorial Identity Store"](#).

Use this method if you are already building secure applications with JDeveloper and have populated `system-jazn-data.xml` with user data of your own.

Once you've completed this preliminary step, you're ready to move on to the actual authentication work for your application.

## Step 1: Creating a Login Page

In this step you'll create a login page that accepts user credentials and allows access to secured pages ([Figure 8-2](#)). Even non-authenticated users will be able to see this page.

**Figure 8-2 The Login Page**

### Login

Enter your user name and password to log in:

Name:

Password:

1. In the Applications Navigator, right-click **ViewController**, and select **New**.
2. In the categories pane, under Web Tier, click **JSP**.

We've chosen to implement the login page as a *standard JSP* rather than a JavaServer Faces (JSF) page to bypass the complexity of the Faces Page Lifecycle. In other applications the login page may require more advanced features such as skinning or portlets and you can find more information about Faces-based login pages in the *Oracle WebCenter Framework Developer's Guide*.

3. Under Items, click **JSP**, and then **OK**.
4. Click **Next** to skip the Welcome page.
5. In the File Name field, enter: `Login.jspx`
6. For Type, select **JSP Document (\*.jspx)**.  
This creates an XML representation of a JSP page (.jspx). You could create the login page as a JSP page in other applications, if you wish.
7. Click **Next**, and then **Next** again to skip the error page options.
8. Make *no selection* on the Tag Libraries page. If necessary, use the double arrows on the Tag Libraries page to remove any libraries from the **Selected Libraries** pane.
9. Click **Next**.
10. Click **Finish** to display `Login.jspx`.
11. Click the **Source** tab.

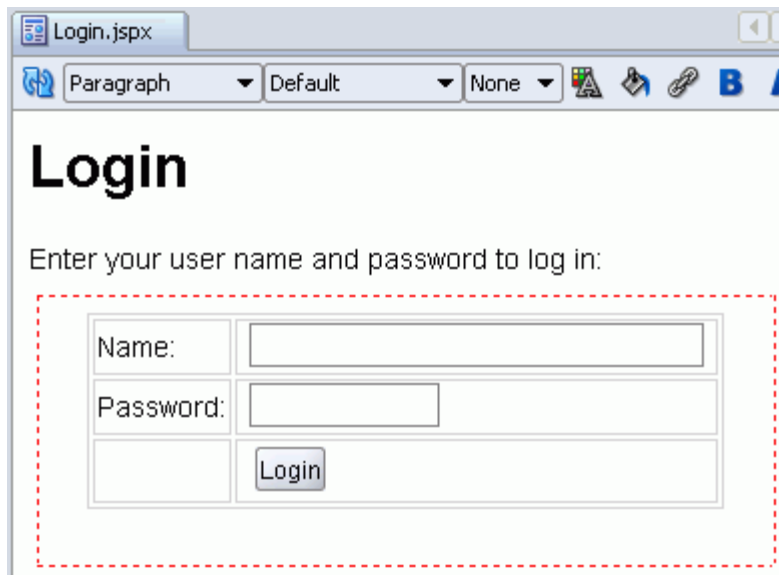
Let's add some code to the body tags that will display a login form.

12. Replace the empty body tags `<body></body>` with the code shown in [Example 8-1](#).

#### **Example 8-1 Login Form Code**

```
<body>
  <h1>Login</h1>
  <p>Enter your user name and password to log in:</p>
  <form action='j_security_check' method='post'>
    <table align="center">
      <tr> <td>Name:</td>
        <td> <input type='text' name='j_username'></input> </td>
      </tr>
      <tr>
        <td>Password:</td>
        <td> <input type='password' name='j_password' size='8'></input> </td>
      </tr>
      <tr> <td></td>
        <td> <input type='submit' value='Login'></input> </td>
      </tr>
    </table>
    <br></br>
  </form>
</body>
```

13. Click the **Design** tab to view the login form ([Figure 8-4](#)).

**Figure 8–3 Login.jspx With Login Form**

The login page uses the standard J2EE security container login method `j_security_check` to validate user credentials. This security check method appears on the `<form>` element. The form itself contains two input fields, one to accept the user name and the other for the password. Values entered into these fields get assigned to the container's login bean attributes `j_username` and `j_password`, respectively.

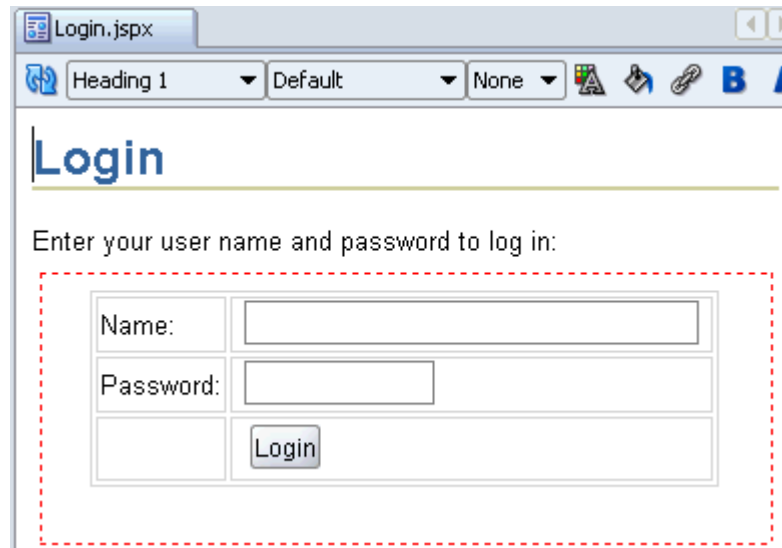
Login Form	Name/Value
Form Action	<code>j_security_check</code>
Name (Text Field)	<code>j_username</code>
Password (Text Field)	<code>j_password</code>
Login (Submit Button)	<code>submit</code>

14. Now, let's apply a style sheet to the login page. From the Components Palette pulldown menu, choose **CSS**.
15. Drag **JDeveloper**, and drop it on the page.

The login page refreshes with the new style sheet applied as shown in [Figure 8–4](#).



Figure 8-4 Login.jspx With JDeveloper Style Sheet

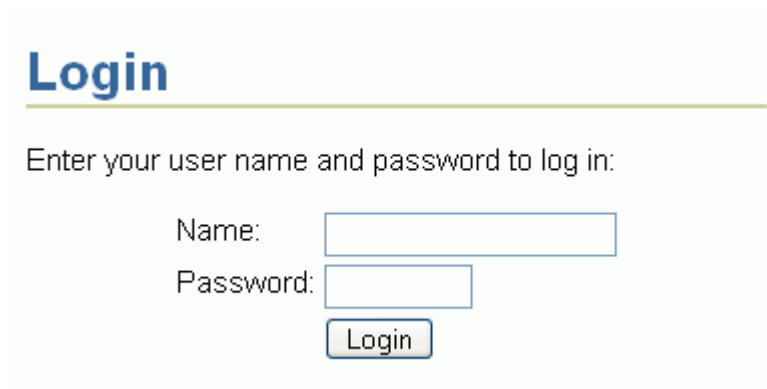


Let's run Login.jspx to see the page in a browser.

16. Click the **Save All** icon in the JDeveloper toolbar.
17. Right-click **Login.jspx**, and select **Run**.

When the page appears in your browser you should see the login form shown in [Figure 8-5](#).

Figure 8-5 Final Login.jspx



18. Close the browser, and return to JDeveloper.

Now let's set Oracle ADF Security options for the tutorial application.

## Step 2: Configuring ADF Security Settings

In this step you'll use the Oracle ADF Security Wizard to configure authentication settings for the tutorial application. The options that you select are recorded in `web.xml` or `orion-application.xml`. Here's a quick overview of what you'll do:

- Enable Oracle ADF authentication.
- Choose the lightweight XML resource provider for user authentication.

- Specify the protocol for authentication as Form-Based.
- Grant authenticated users (ValidUsers) access to the `adfAuthentication` servlet.

You'll find more information in *Oracle WebCenter Framework Developer's Guide*.

Let's start the ADF Security Wizard now:

1. In the Applications Navigator, select **ViewController**.
2. From the **Tools** menu, choose **ADF Security Wizard**.  
The ADF Security wizard will guide you through the configuration process.
3. Click **Next** to skip the Welcome page.
4. Select **Enforce Authorization** as shown in [Figure 8-6](#).

This option configures ADF Security functionality by configuring the `adfAuthentication` servlet used for authentication and other servlets and filters required to enforce authorization policies (the filters enforce for the checking of the current user's permissions for the requested page).

**Figure 8-6 ADF Security Wizard - Enabling Authentication**

Configure ADF Security For A Web Application  
 Enforce Authorization  
 Redirect Upon Successful Authentication  
 URL:   
 Generate Default

If you want users logging into your application to see a particular page after authentication, you would choose **Redirect After Successful Authentication** and name the required page. However, you're not going to configure the tutorial application like this. Instead, you'll use the default behavior, which sends authenticated users back to the page they tried to access prior to authentication.

5. Click **Next** to move on to the next page of the wizard.
6. Choose **Lightweight XML Provider** ([Figure 8-7](#)).

**Figure 8-7 ADF Security Wizard - Choosing a JAAS Provider**

Lightweight XML Provider  
 Suitable for small scale applications and for testing purposes.

Oracle ADF Security authenticates users against a given resource provider. For this tutorial you'll use the lightweight resource provider `system-jazn-data.xml` that you set up at the beginning of the lesson.

7. Click **Next** to display the next page of the wizard.
8. Set JAAS Mode to **doAsPrivileged** ([Figure 8-8](#)).

Oracle ADF Security requires this option to be set.

**Figure 8–8 ADF Security Wizard - Configuring XML Settings**

Location: OC4J Default Repository

Default Realm: jazn.com

JAAS Mode: doAsPrivileged

The default realm is *jazn.com*. Note that for this release, Oracle ADF security only reads permissions at the system-level (JAZN file) and not at the application-level.

9. Click **Next**.
10. On the Login page, choose **Form-Based Authentication** as shown in [Figure 8–9](#). This specifies that the tutorial application will use a form to facilitate authentication.

There is no need to generate default pages for the login form and login error message (`login.html` and `error.html`) because you'll be using the login form in `Login.jspx`.

11. For Login Page, enter: `Login.jspx`
12. For Error Page, you can also enter: `Login.jspx`

**Figure 8–9 ADF Security Wizard - Configuring Form-Based Authentication**

Form-Based Authentication

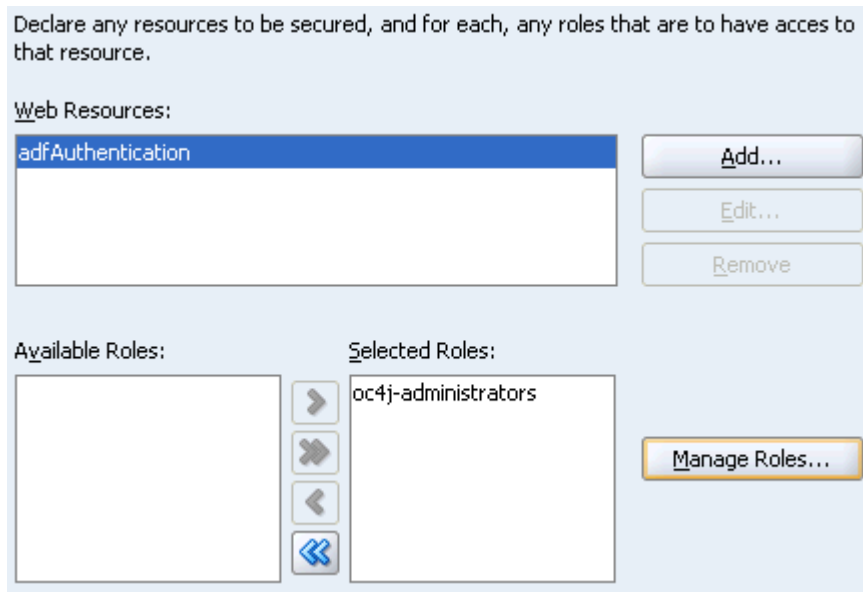
Login Page: Login.jspx

Error Page: Login.jspx

Generate Default

Although it's fairly simple to create a separate error page, for this tutorial you'll use the same page. If you do want to create an error page (say, `LoginError.jspx`), create a page that is identical to `Login.jspx` and add an error message that indicates authentication failure.

13. Click **Next** to display the final page of the wizard - *Resources* ([Figure 8–10](#)). This page defines resources within your application that should be secured, and specifies which J2EE security roles can access each resource.

**Figure 8–10 ADF Security Wizard - Securing the `adfAuthentication` Servlet**

The `adfAuthentication` resource (the authentication servlet) is defined for you. This servlet acts as a known endpoint for a login URL. As the servlet itself is secured by a J2EE security constraint, it causes a redirect to the login page when a user attempts to access it without a current active session.

You cannot edit or delete this resource, but you can specify the set of roles that may access this resource.

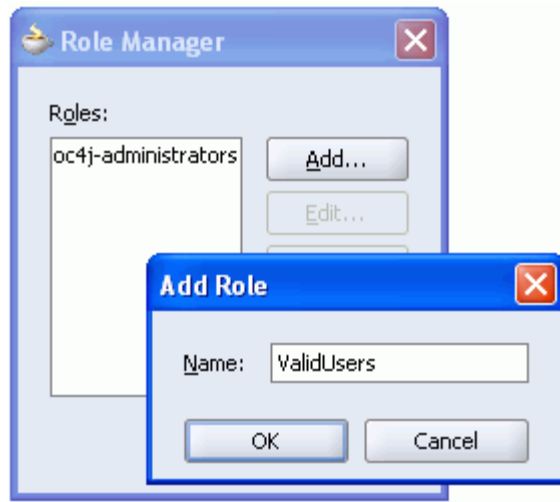
To allow any valid user access to the `adfAuthentication` resource, you'll need to create a J2EE role, which you'll name `ValidUsers`, and grant access to this role. Let's do that now.

**14.** Click **Manage Roles**.

**15.** Click **Add**, and enter the name `ValidUsers` (Figure 8–11).

Later on, you'll map this J2EE role to an identity store role named `users` (defined in `system-jazn-data.xml`, the resource provider that you set up at the beginning of the lesson). The `users` role maintains a list of every valid user.

From a security perspective, allocating permissions to this role effectively defines an authenticated *Public* resource. That is, it would be available to all users without needing a specific permission to be defined.

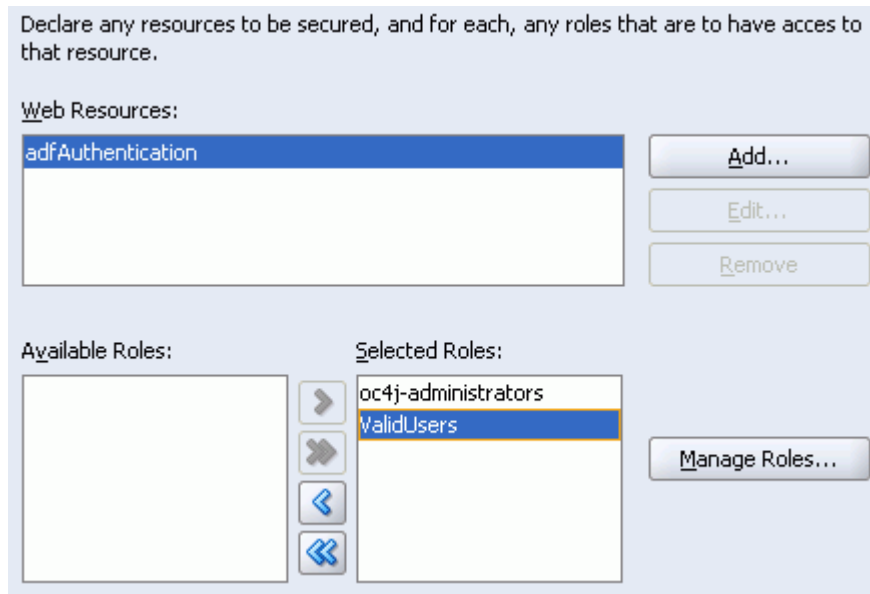
**Figure 8–11 ADF Security Wizard - Adding a J2EE Security Role**

16. Click **OK**.

17. Click **Close**.

The ValidUsers role should appear in the list of Available Roles.

18. Click the double arrow (**Add All**) to move everything in the **Available Roles** list to the **Selected Roles** list (Figure 8–12).

**Figure 8–12 ADF Security Wizard - Granting Access to the adfAuthentication Resource**

This completes the ADF Security wizard settings.

19. Click **Next**, and then **Finish**.

20. Click the **Save All** icon in the JDeveloper toolbar.

Before going on to the next step of this tutorial, take a look at the changes you've just made to `web.xml` and `orion-application.xml`. You'll find these files in the Applications Navigator, under:

- ViewController, Web Content, WEB-INF, web.xml
- ViewController, Application Sources, META-INF, orion-application.xml

Double-click the file name to display the file in the XML Editor.

**Tip:** You can expand any tab to fill the JDeveloper window by double-clicking the tab. Double-clicking again, toggles the feature off.

One way to review file changes is through the History Tool. If you click the History tab, you'll see the additions highlighted down the right-hand side. You can click change highlights individually or use the **Go to Next Difference** icon to scroll through the additions one by one. You'll find more information about these entries in the Security Chapter of the *Oracle WebCenter Framework Developer's Guide*.

Now let's create a welcome page for the tutorial application.

### Step 3: Creating a Welcome Page

In this step you'll create a welcome page, which will serve as the starting page for the tutorial application. When non-authenticated users display the welcome page, they'll see a simple login link that directs them to a login page for authentication. This is the public view of the page (Figure 8-13).

**Figure 8-13** Your Welcome Page - Public View



Authenticated users will see different information on the welcome page. After logging in, authenticated users get redirected back to the welcome page, but this time they'll see links to secured pages and a logout link as shown in Figure 8-14.

**Figure 8-14** Welcome Page - Secured View



Let's create the welcome page now:

1. In the Applications Navigator, right-click **ViewController**, and select **New**.
2. In the categories pane, under Web Tier, click **JSF** to create a Java Server Faces page.
3. Under Items, click **JSF JSP** and click **OK**.
4. Click **Next** to skip the Welcome page.

5. In the File Name field, enter: `Welcome.jspx`
6. For Type, make sure **JSP Document (\*.jspx)** is selected and then click **Next**.
7. You'll be adding some back end logic to this page, so you'll need managed bean. Click the radio button **Automatically Expose UI Components in a New Managed Bean**.

You can keep the defaults provided for Name, Class, and Package.

8. Click **Next**.
9. Ensure that the following libraries are in the **Selected Libraries** pane:

- ADF Faces Components 10\_1\_3\_2\_0
- ADF Faces HTML 10\_1\_3\_2\_0
- ADF Portlet Components 10\_1\_3\_2\_0
- Customizable Components Core 10\_1\_3\_2
- JSF Core 1.0
- JSF HTML 1.0

10. Click **Next**.

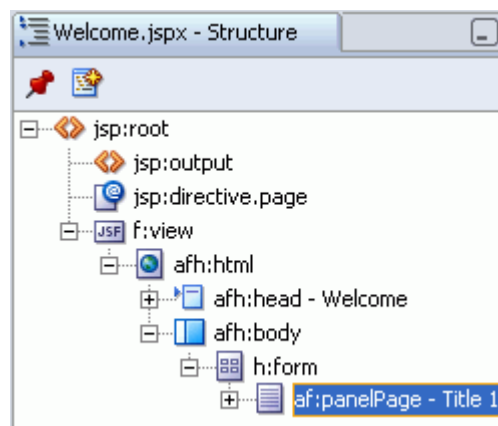
11. Earlier, we applied the JDeveloper style sheet to the login page. Let's apply the JDeveloper style sheet to the welcome page too. Click **Add**, double-click the **css** folder, and then choose **jdeveloper.css**.

12. Click **Finish**.

An empty page named `Welcome.jspx` displays in Design view.

13. Let's use an ADF Faces *PanelPage* component to present content on the welcome page:
  - a. From the Components Palette pulldown, choose **ADF Faces Core**.
  - b. Select the **PanelPage** option, and then drag and drop it on to the page. Use the Structure window to verify that the *PanelPage* component gets placed inside the `h:form` tag, as shown in [Figure 8-15](#).

**Figure 8-15** *Welcome.jspx* - ADF Faces *panelPage* Component

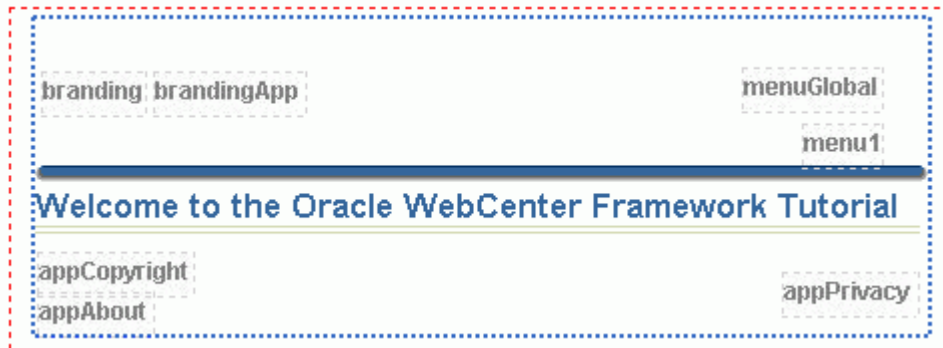


- c. In the Structure window, double-click **af:panelPage - Title1** to display the *PanelPage* Properties dialog.

- d. For the Title, enter: Welcome to the Oracle WebCenter Framework Tutorial
- e. Click **OK**.

Now your welcome page should look something like [Figure 8–16](#). You don't need to set any other `af:panelPage` properties for this tutorial.

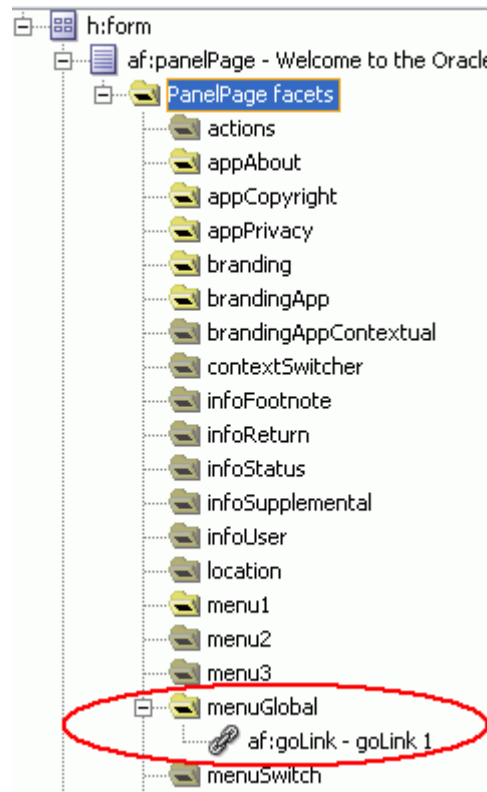
**Figure 8–16** *Welcome.jsx with New panelPage Title*



Now let's add some Login/Logout links to the area named `menuGlobal`—one of the `PanelPage` facets.

- 14. You'll use an `af:goLink`, another ADF Faces Core component, for the login link:
  - a. From the Component Palette pulldown, choose **ADF Faces Core**.
  - b. Select the **GoLink** option, and then drag and drop it on top of the area named `menuGlobal` in the top right-hand corner. Use the Structure window to verify that it is placed inside the `menuGlobal` facet as shown in [Figure 8–17](#).



**Figure 8–17 Welcome.jspx - ADF Faces GoLink Component**

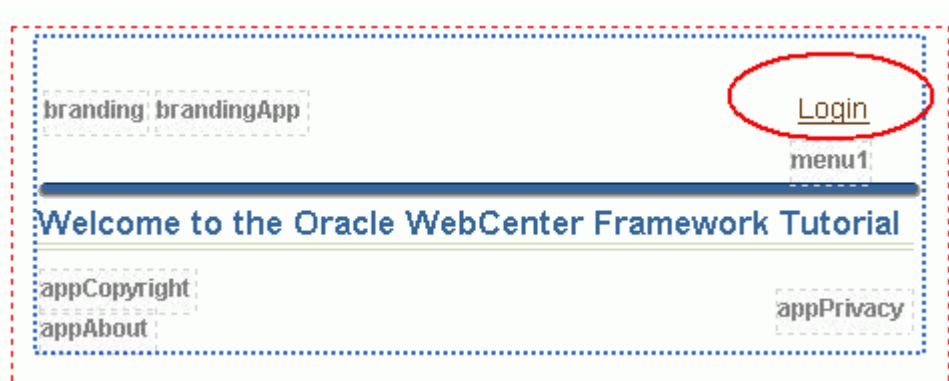
- c. In the Structure window, double-click **af:goLink - goLink1** to display its Properties dialog.
- d. For the link Text, enter: Login
- e. In the Destination field, enter:

```
/adfAuthentication?success_url=faces/Welcome.jspx
```

The `adfAuthentication` servlet will prompt users to login. The parameter `success_url` specifies which page to display after a successful login, which for this tutorial will be the Welcome page.

- f. Click **OK**.

You should see a Login hyperlink like the one in [Figure 8–18](#).

**Figure 8–18 Login Link on the Welcome Page**

The page content you see now (Figure 8–18) is the *public view* of the Welcome page. A little bit later on you'll add some page navigation buttons but you'll add some code that will hide them from *unauthenticated* users.

Similarly, you'll need to add some code that will hide the Login link from *authenticated* users. Clearly, there is no need to display a login link to users who are already logged in! So let's add a backing bean that determines whether the current user is authenticated and bind it to the login link.

15. First, double-click **Welcome.java** to open the file in the Editor.  
You'll find this file under **ViewController, Application Sources, view.backing**.
16. Add some code that determines whether or not the current user is logged in, see highlighted code shown in Figure 8–19.

**Figure 8–19 Welcome.java**

```

package view.backing;
import javax.faces.component.html.HtmlForm;
import oracle.adf.share.ADFContext;
import oracle.adf.view.faces.component.core.layout.CorePanelPage;
import oracle.adf.view.faces.component.core.nav.CoreGoLink;
import oracle.adf.view.faces.component.html.HtmlBody;
import oracle.adf.view.faces.component.html.HtmlHead;
import oracle.adf.view.faces.component.html.HtmlHtml;

public class Welcome {
    private HtmlHtml html1;
    private HtmlHead head1;
    private HtmlBody body1;
    private HtmlForm form1;
    private CorePanelPage panelPage1;
    private CoreGoLink goLink1;
    private boolean authenticated;

    public void setHtml1(HtmlHtml html1) {...}

    public HtmlHtml getHtml1() {...}

    public CoreGoLink getGoLink1() {...}

    public boolean isAuthenticated() {
        authenticated =
            ADFContext.getCurrent().getSecurityContext().isAuthenticated();
        return authenticated;
    }
}

```

You can copy and paste the bold code shown here to the appropriate section of Welcome.java:

```

package view.backing;

import javax.faces.component.html.HtmlForm;

import oracle.adf.share.ADFContext;
import oracle.adf.view.faces.component.core.layout.CorePanelPage;
import ...

public class Welcome {
    ...
    private CoreGoLink goLink1;
    private boolean authenticated;
    ...

    public CoreGoLink getGoLink1()
    {return goLink1;
    }
    public boolean isAuthenticated()
    {
        authenticated=ADFContext.getCurrent().getSecurityContext().isAuthenticated();
        return authenticated;
    }
}

```

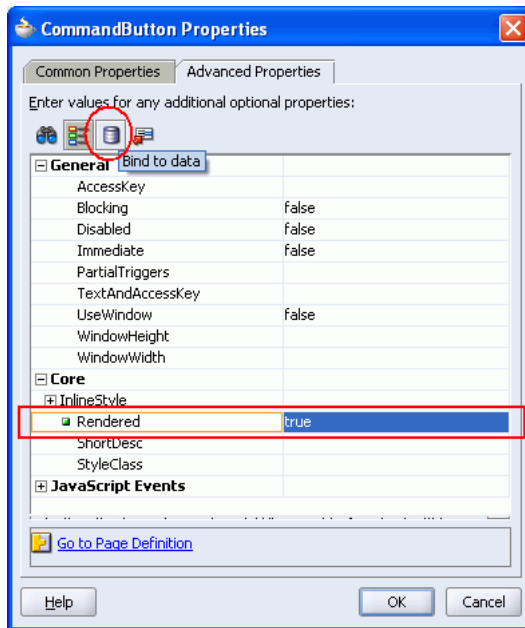
The boolean property of the backing bean is TRUE if the current user is logged in and FALSE if he is not. You can use this property to show /hide the links on the Welcome page based on whether the user is logged in.

17. Before moving on, let's check that the updates to Welcome.java compile correctly. Right-click Welcome.java, and choose **Make**.

You should see a Successful compilation message in the Messages Log window.

18. Now bind this code to the Login link. Remember, we want to display the Login link to *unauthenticated* users only:
  - a. In the Applications Navigator, double-click **Welcome.jspx** to display the page in the Visual Editor.
  - b. In the Structure window, double-click **af:goLink - Login** to open the GoLink Properties dialog.
  - c. Click the **Advanced Properties** tab, and select the **Rendered** property.
  - d. Click the **Bind to Data** icon in the toolbar (see [Figure 8–20](#)).

Figure 8–20 Command Button Rendered Property



- e. Expand **JSF Managed Beans** and drill into the backing bean for the Welcome page **backing\_Welcome**. You should see the **authenticated** attribute listed here.
  - f. Double-click **authenticated**. The expression `{backing_Welcome.authenticated}` should appear in the Expression field.
  - g. We want to display a Login link to users who are *not* authenticated. The **Authenticated** property will be **TRUE** if the user is logged in, so you must negate the expression to show the link only when the user is not logged in. Add the **!** operator to the expression like this: `{!backing_Welcome.authenticated}`
  - h. Click **OK**.  
The **Rendered** property should now display the expression `{!backing_Welcome.authenticated}`.
  - i. Click **OK** again to close the **GoLink Properties** dialog.
19. Now let's add a logout link that's displayed only to *authenticated* users:
- a. From the **Component Palette** pull-down, choose **ADF Faces Core**.
  - b. Select the **GoLink** option, and then drag and drop it below the Login link (inside the **menuGlobal** facet). You can use the **Structure** window to verify this.
  - c. In the **Structure** window, double-click **af:goLink - goLink 1** to open the **GoLink Properties** dialog.
  - d. For the link **Text**, enter: Logout
  - e. In the **Destination** field, enter:  
`/adfAuthentication?logout=true&end_url=faces/Welcome.jspx`

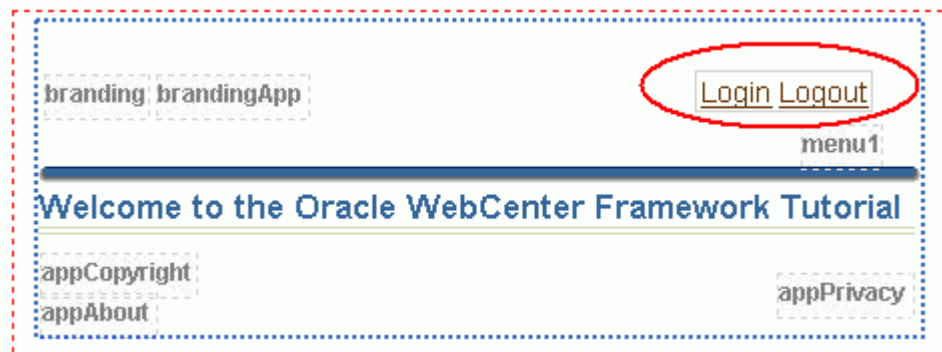
The `adfAuthentication` servlet will prompt users to log out and the parameter `end_url` specifies which page to display once the user is logged out.

- f. Now let's hide the Logout link from unauthenticated users. Click the **Advanced Properties** tab, and select the **Rendered** property.
- g. Click the **Bind to Data** icon in the toolbar.
- h. To display this link to authenticated users, enter into the expression field:
 

```
#{backing_Welcome.authenticated}
```
- i. Click **OK**.
- j. Click **OK** again to close the properties dialog.

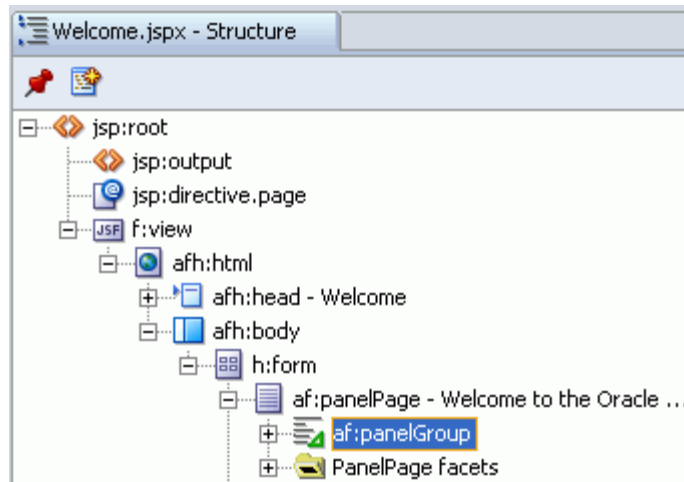
You should see a Logout hyperlink on the Welcome page (Figure 8-21).

**Figure 8-21 Logout Link on Welcome Page**



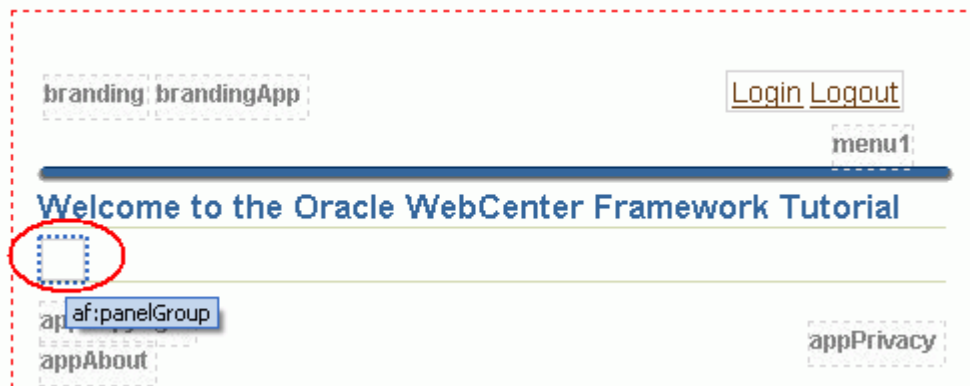
Finally, let's add some command buttons to the Welcome page that allow authenticated users to navigate to the pages in your tutorial (`MyPage.jspx`, `MyWeather.jspx`, and `MyContent.jspx`).

20. Let's use an ADF Faces `panelGroup` component to accommodate the page navigation buttons. With this component you can arrange the content horizontally or vertically.
  - a. From the Component Palette pulldown, choose **ADF Faces Core**.
  - b. Select the **PanelGroup** option, and then drag and drop it on top of the `af:panelPage` component. Again, use the Structure window to verify that it is placed inside the `af:panelPage` tag as shown in Figure 8-22.

**Figure 8–22** *Welcome.jspx - ADF Faces PanelGroup Component*

- c. In the Structure window, double-click **af:panelGroup** to display its Properties dialog.
- d. For Layout, choose **horizontal**.
- e. Click **OK**.

You should see an empty panel under the page title ([Figure 8–23](#)).

**Figure 8–23** *Welcome.jspx - Empty panelGroup Component*

21. Before you can add a command button that navigates from Welcome.jspx and to MyPage.jspx, you'll need to create a JSF navigation rule between the two pages. Then you may drop an ADF command button onto the page, and add some button code that ensures the button displays to authorized users only.
  - a. First, let's define the navigation rule between Welcome.jspx and MyPage.jspx. You do this in the configuration file `faces-config.xml`. In the Applications Navigator, double-click **faces-config.xml** (under ViewController, Web Content, WEB-INF).
  - b. Click the **Diagram** tab.
  - c. Drag **Welcome.jspx** from the Applications Navigator to the empty diagram, and then do the same for **MyPage.jspx** ([Figure 8–24](#)).

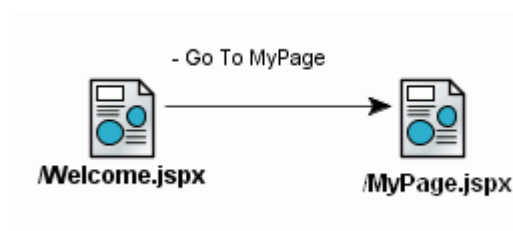
**Figure 8–24 JSF Navigation Diagram**

Notice that the Component Palette automatically displays the *JSF Navigation Modeler* components.

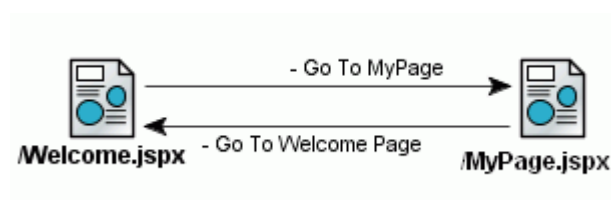
- d. From the Component Palette, choose **JSF Navigation Case** to activate it. You'll use this component to add a navigation rule.
- e. On the diagram, click the icon for the source page (**Welcome.jspx**) and then click the icon for the destination page (**MyPage.jspx**).

JDeveloper draws the navigation rule on the diagram as a solid line ending with an arrow between the two pages,

- f. By default, the link outcome says `-success`. Click this text, and change it to something more explanatory, like `Go To MyPage` (Figure 8–25).

**Figure 8–25 JSF Navigation Diagram With Rule**

- g. Now let's follow similar steps to create a navigation rule from **MyPage.jspx** to **Welcome.jspx**. First, choose **JSF Navigation Case** from the Component Palette. Next, click **MyPage.jspx** (on the diagram) followed by **Welcome.jspx**. Finally, click the default `-success` text and change it to `Go To Welcome Page` (Figure 8–26).

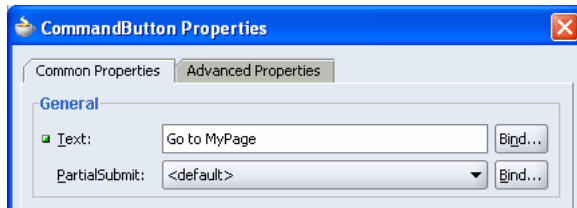
**Figure 8–26 JSF Navigation Diagram With Two Rules**

Now that navigation rules exist between the two pages, you can add a Command button on the Welcome page for user navigation to MyPage.

- h. In the Applications Navigator, double-click **Welcome.jspx** to display it in the Visual Editor.
- i. Choose **ADF Faces Core** from the Component Palette pulldown menu.

- j. Select the **CommandButton** option, and then drag and drop it on top of `af:panelGroup`. You can use the Structure window to verify this.
- k. Right-click the new button (`commandButton 1`), and then choose **Properties**.
- l. For the button Text, enter `Go To MyPage` as shown in [Figure 8–27](#), and click **OK**.

**Figure 8–27** *Welcome Page.jspx - Configuring Page Navigation Button*



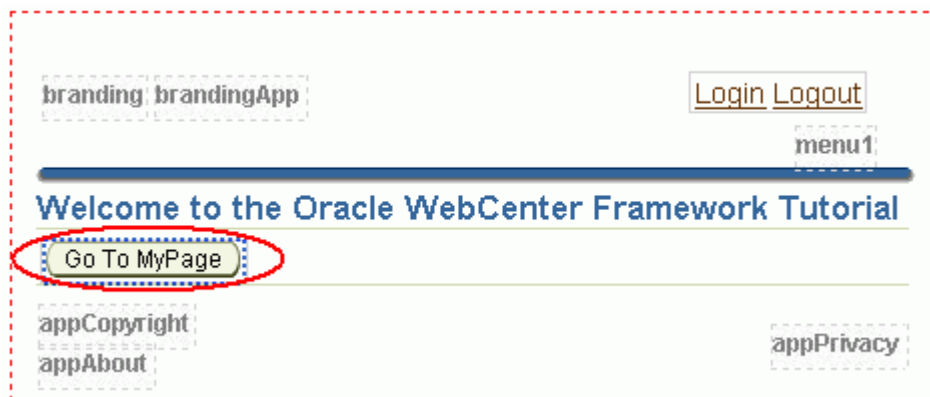
- m. You define what the button actually does through the button's Action property. You'll need to display the Property Inspector to do this, so select **View, Property Inspector** from the JDeveloper menu.
- n. In the Property Inspector, select **Action**, and then choose **Go To MyPage** from the drop down list.

Now, let's hide the button from unauthenticated users—just like you did for the Logout link. In addition, we'll add some code that will hide the button if the logged-in user is not authorized to view `MyPage.jspx`. This time we'll use the Property Inspector to enter the expression.

- o. Select the **Rendered** property. In the Property Inspector, this property is located under the **Core** section.
- p. Now click the **Bind to data** icon in the Inspector's toolbar.
- q. Enter the expression: `#{backing_Welcome.authenticated && bindings.permissionInfo['MyPagePageDef'].allowsView}`  
Check there are no extra spaces at the beginning or end of this expression.
- r. Click **OK**.

In JDeveloper, your Welcome page should look like [Figure 8–28](#).

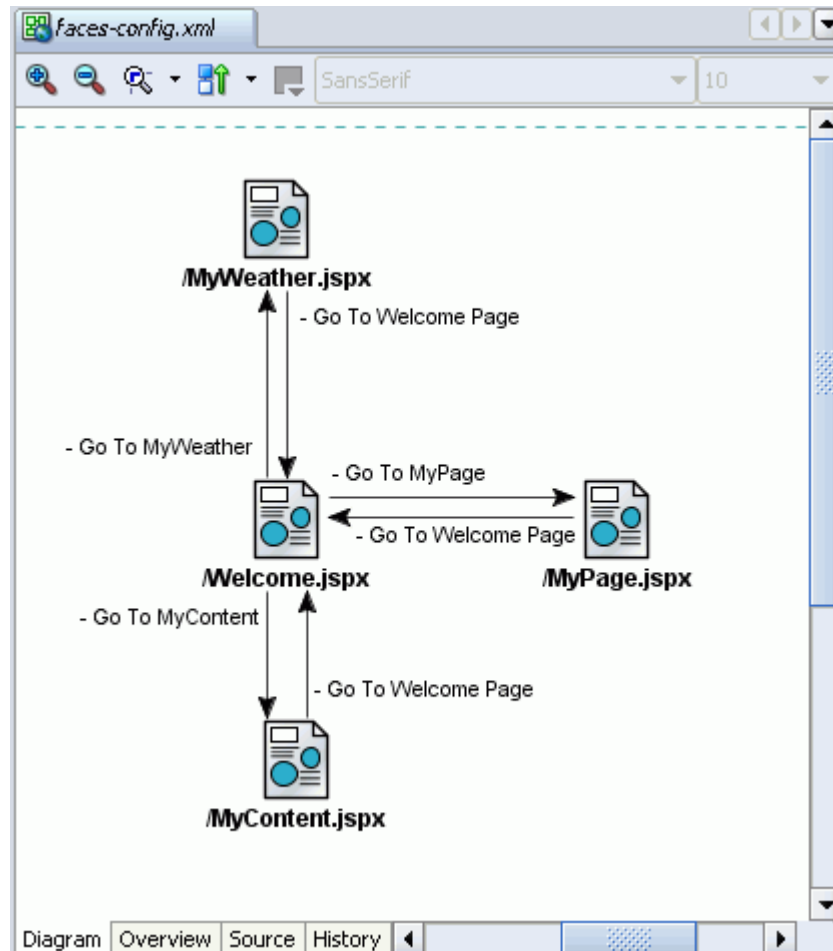
**Figure 8–28** *MyPage Navigation Button on Welcome Page*





22. Now add two more buttons to the Welcome page that will enable authorized user navigation to MyWeather.jsx and MyContent.jsx. Repeat step 21 *very* carefully.
- First, add MyWeather.jsx and MyContent.jsx to the navigation diagram. When complete, check that your navigation diagram looks similar to Figure 8–29.

**Figure 8–29 Completed JSF Navigation Diagram**



- Now, add two more navigation buttons to Welcome.jsx, and set the button **Text**, **Action**, and **Rendered** properties.

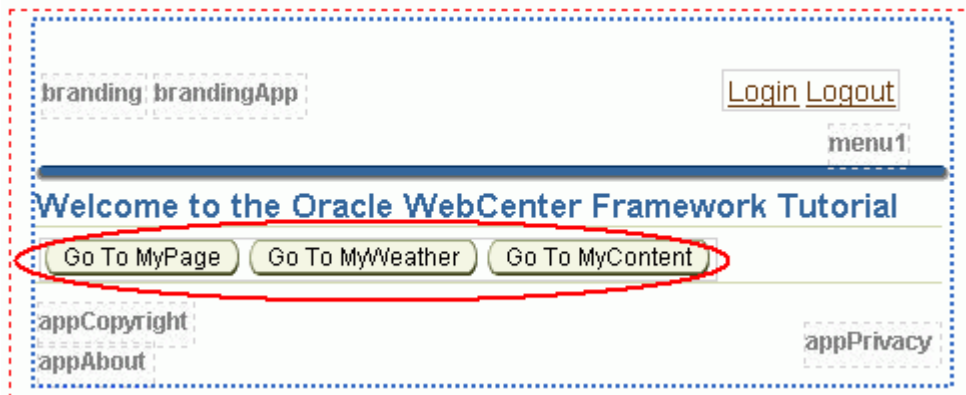
When you edit the **Text**, **Action**, and **Rendered** properties for these navigation buttons, remember to substitute the appropriate page names (as shown in Table 8–1) and check for extra spaces at the beginning/end of the Rendered expression.

**Table 8–1 Command Button Properties**

Button Text	Action	Rendered
Go To MyWeather	Go To MyWeather	<code>#{backing_Welcome.authenticated &amp;&amp; bindings.permissionInfo['MyWeatherPageDef'].allowsView}</code>
Go To MyContent	Go To MyContent	<code>#{backing_Welcome.authenticated &amp;&amp; bindings.permissionInfo['MyContentPageDef'].allowsView}</code>

Your Welcome page should look something like [Figure 8–30](#).

**Figure 8–30** Final Welcome Page



23. Click the **Save All** icon in the JDeveloper toolbar.

Let's run the Welcome page to see whether the links and buttons you've just added display as expected.

24. Right-click **Welcome.jsx**, and choose **Run**.

In the browser, you should see the Welcome page with a Login link as shown in [Figure 8–31](#). Without authentication, the logout link and page navigation buttons should not display.

If your page does not look like this, check the Rendered expression for any links or buttons which do not display as expected.

**Figure 8–31** Welcome Page - Public View



25. Close the browser, and return to JDeveloper.

In the next step you'll authorize access privileges for each page in your application (Welcome.jsx, MyPage.jsx, MyWeather.jsx, and MyContent.jsx).

## Step 4: Securing Pages

In this step you'll secure (restrict access to) the pages in this tutorial application: Welcome.jsx, MyPage.jsx, MyWeather.jsx, MyContent.jsx. You'll restrict page access to the role members defined in the identity store (see [Prerequisites](#)) and dictate the actions that role members can perform on the page. Secured pages also need logout links, so you'll add logout links at the top of each page too.

Let's start with the Welcome page. This page already has a logout link, but you still need to authorize access to the page and grant permissible actions. This configuration takes place in the page's definition file (WelcomePageDef.xml). Let's get started.

1. Right-click **Welcome.jspx** in the Applications Navigator.
2. Choose **Go to Page Definition**.

If the Page Definition doesn't exist yet, click **Yes** to create one for Welcome.jspx.

3. In the Structure window, right-click **WelcomePageDef**, and choose **Edit Authorization**.

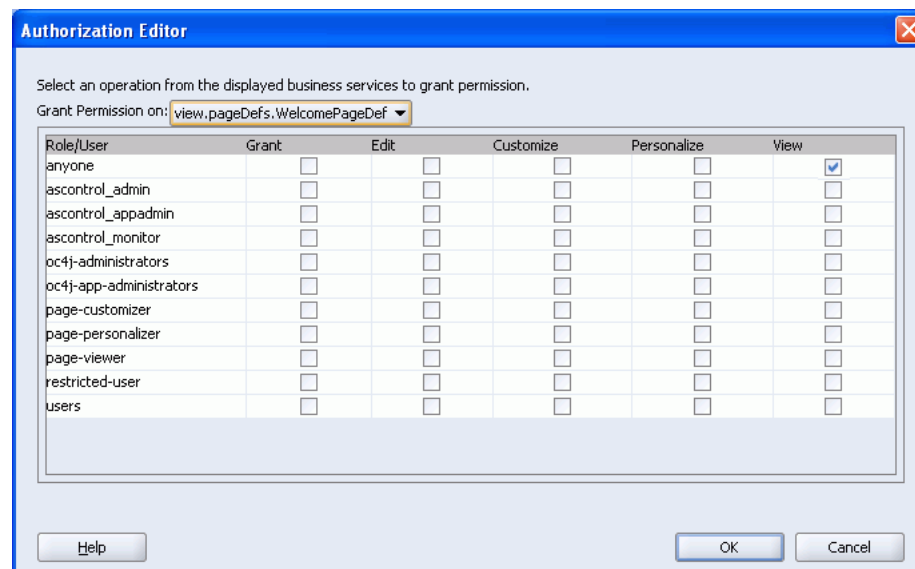
The Authorization Editor should list the identity store roles shown in [Figure 8-32](#).

If these roles do not display, check you've copied the file `system-jazn-data.xml` from the embedded OC4J directory `JDEVHOME\jdev\system\oracle.j2ee.10.1.3.xx.xx\embedded-oc4j\config` to the directory `JDEVHOME\j2ee\home\config`. This is one of the [Prerequisites](#) for this chapter.

The Authorization Editor also enables you to choose which page actions each role may perform:

- **Grant** - Users may administer (grant/revoke) page permissions.
  - **Edit** - Users may edit content displayed on the page. The Edit action is not applicable for this release.
  - **Customize** - Users may modify the page. If a user is not granted this permission, he will not be able to modify the page.
  - **Personalize** - Users may personalize portlets on the page. If a user is not granted this permission, links or buttons that put page portlets into personalization mode are not displayed.
  - **View** - Users may view the page. If a user is not granted this permission he will see an authorization error.
4. Anyone can view Welcome.jspx. Select the **View** check box against the **anyone** role, as shown in [Figure 8-32](#).

**Figure 8-32** *Welcome.jspx - Authorization Editor*



5. Click **OK**.

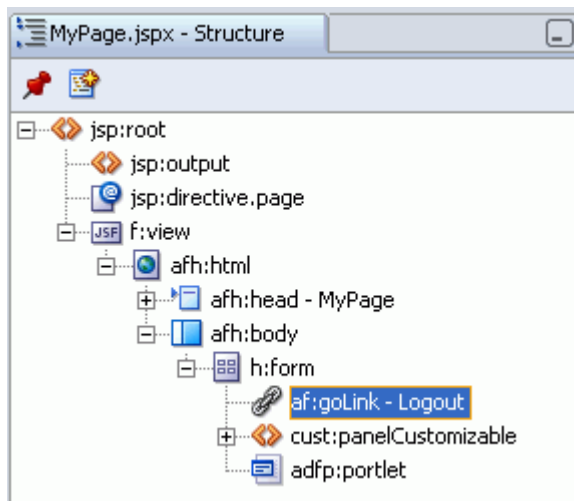
Changes made through the Authorization Editor are saved in the embedded OC4J's `system-jazn-data.xml` file, for immediate testing, and simultaneously saved to `app-jazn-data.xml` in the application's `.adf/META-INF` directory. The `app-jazn-data.xml` file packages *application specific* security policies along with the application itself, to facilitate the deployment of secure WebCenter applications. You'll learn more about `app-jazn-data.xml` in [Chapter 9, "Deploying Your WebCenter Application"](#).

Let's secure `MyPage.jspx` next. First you'll add a logout link, and then you'll edit page authorization details. Instead of creating a new Logout link from scratch, copy the logout link you created for the Welcome page.

6. In the Applications Navigator, select **Welcome.jspx**.
7. In the Structure window, right-click **af:goLink - Logout**, and choose **Copy**.
8. In the Applications Navigator, select **MyPage.jspx**.
9. In the Structure window, select **h:form**, right-click, and select **Paste**.

`af:goLink - Logout` is placed at the bottom of the page by default. Drag it above `cust:panelCustomizable` to move it to the top of the page ([Figure 8-33](#)).

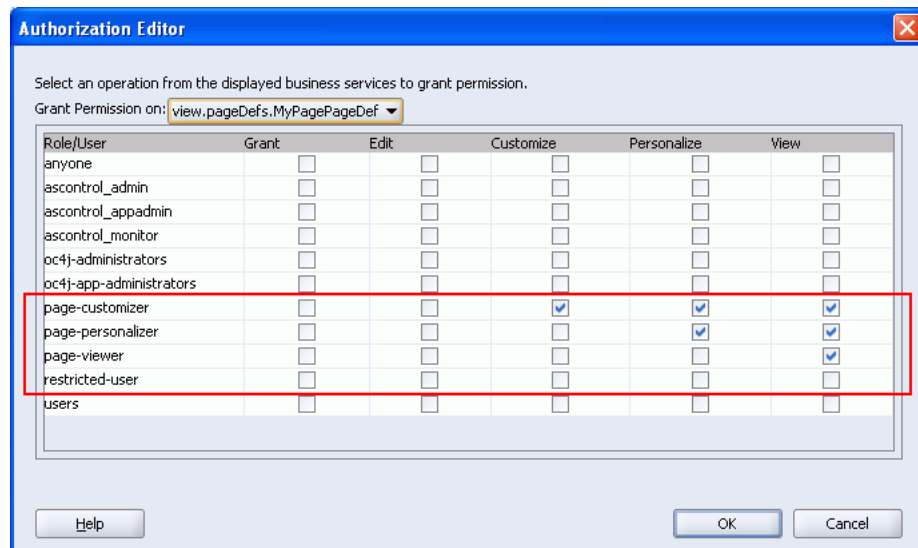
**Figure 8-33** *MyPage.jspx - Logout Link in Structure Window*



Now let's authorize access to this page and specify permissible actions on a role by role basis. As before, this configuration takes place in the page's definition file (`MyPagePageDef.xml`).

10. Right-click **MyPage.jspx** in the Applications Navigator.
11. Choose **Go to Page Definition**.
12. In the Structure window, right-click **MyPagePageDef**, and choose **Edit Authorization**.
13. Use the Authorization Editor to grant user permissions on `MyPage.jspx`. Select the checkboxes shown in [Figure 8-34](#).

Figure 8–34 MyPage.jspx - Authorization Editor



Let's take a closer look at these settings. When a user assigned with the *page-customizer* role logs in to the tutorial application (such as *Harvey*), that user may view, personalize, and customize aspects of MyPage.jspx, but may not edit page content or grant page permissions to any other users. Logged-in users with the *page-viewer* role (such as *Singh*) are more restricted; those users can view MyPage.jspx but no other page actions are allowed.

Notice that users with the *restricted-user* role (such as *King*) are not authorized to view the page at all.

- Click **OK** to save these selections.

Now that MyPage.jspx is secure, repeat exactly the same steps for MyWeather and MyContent.

- To secure MyWeather and MyContent, repeat steps 8 through 14.

After you copy and paste the Logout links, remember to move them to the top of your pages (under the `h:form`).

- Click the **Save All** icon in the JDeveloper toolbar.

All the pages in your application are now secure.

## Step 5: Mapping Security Roles in orion-web.xml

There is one more step to secure your tutorial application. You need to map the J2EE security role you defined using the Oracle ADF Security wizard (`ValidUsers`) to an identity store role defined in your `system-jazn-data.xml` file (`users`).

J2EE Security Role	Identity Store Role
ValidUsers	users

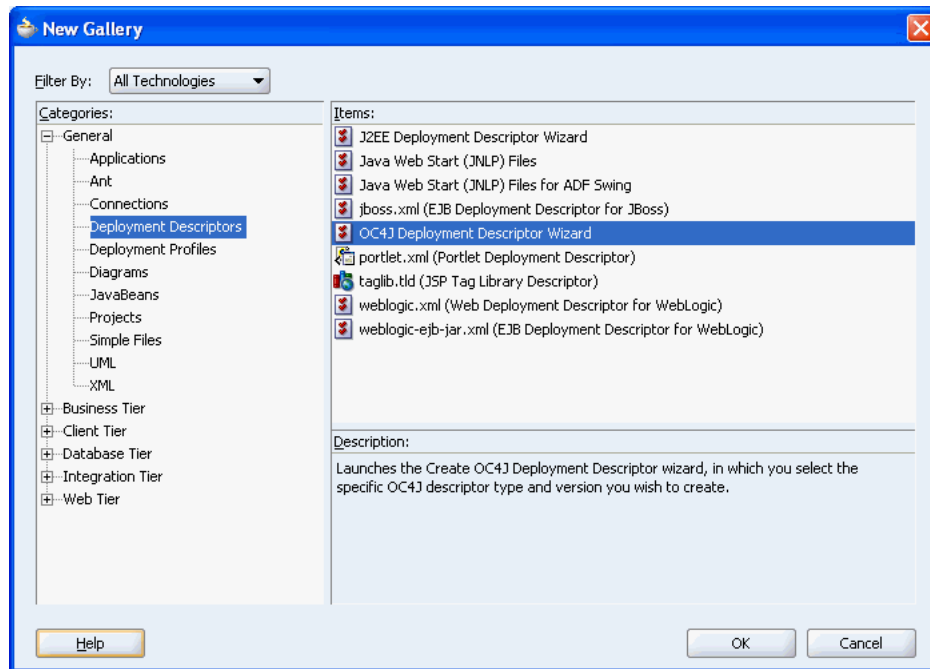
First, you'll create the configuration file where such security role mappings are stored; that is, the tutorial application's OC4J deployment descriptor file (`orion-web.xml`). Let's create the file now.

- In the Applications Navigator, right-click **ViewController**, and select **New**.

2. Select **All Technologies** from the pull-down list at the top.
3. In the panel on the left, expand **General**, and then select **Deployment Descriptors** as shown in [Figure 8–35](#). The panel on the right lists the different types of deployment descriptors that are available. Projects require different deployment descriptors depending on the technologies the project uses and on the type of the target application server.

This tutorial application will be deployed on the preconfigured OC4J server.

**Figure 8–35** *Deployment Descriptor Selection*



4. Select **OC4J Deployment Descriptor Wizard**, and then click **OK** ([Figure 8–35](#)).  
This brings up the OC4J Deployment Descriptor wizard. This wizard lets you select the specific deployment descriptor type and version you require, and then generates the .xml file. For this tutorial you need to select `orion-web.xml 10.0` as the deployment descriptor. Later on you'll use this .xml file to package the tutorial application before deployment to the application server.
5. Click **Next** to move beyond the Welcome page.
6. Select **orion-web.xml** (located at the bottom of the list), and then click **Next**.  
The file name will be greyed out if the file already exists.
7. Select **10.0**, and then click **Finish**.  
After you click Finish, `orion-web.xml` displays in the Applications Navigator, under **ViewController**, **Web Content**, **WEB-INF**.
8. Right-click **orion-web.xml**, and select **Properties** to set some additional deployment options.
9. Select **Security Role Mappings** from the panel on the left.  
This displays a panel on the right where you'll add the following security role mapping:

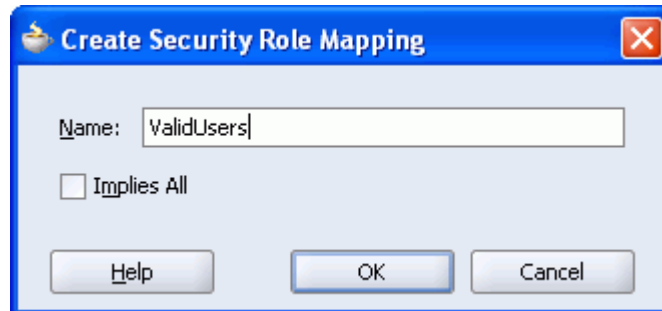
J2EE Security Role	Identity Store Role
ValidUsers	users

10. Create the security role mapping as follows:

a. Click **Add**.

This displays a window, as shown in [Figure 8–36](#), where you'll enter the J2EE security role name—ValidUsers.

**Figure 8–36** Deployment Descriptor - Creating J2EE Security Role Mapping

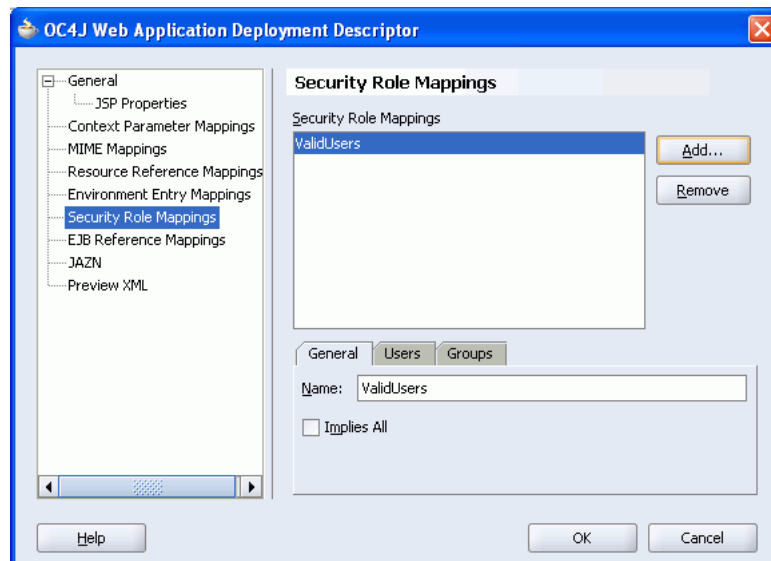


b. For Name, enter the J2EE security role name: ValidUsers

c. Click **OK**.

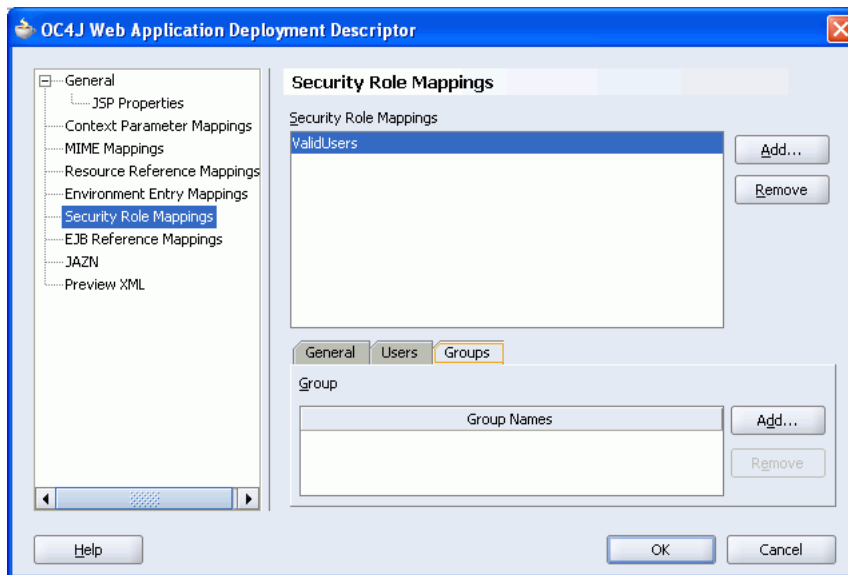
The role name you just entered gets displayed in the mappings panel and also on the **General** tab as shown in [Figure 8–37](#). If you must edit the role name you can do so by editing the **Name** property on the General tab.

**Figure 8–37** Deployment Descriptor - New J2EE Security Role Mapping



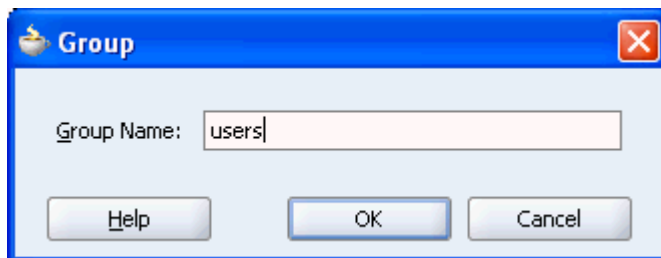
d. Click the **Groups** tab.

Notice that the J2EE security role ValidUsers is highlighted in the mappings panel. This means that you're about to map a group of users to this J2EE security role ([Figure 8–38](#)).

**Figure 8–38** Deployment Descriptor - Security Role Mappings Groups Tab

- e. Click the **Add** button to the right of the Group Names panel.
- f. For Group Name, enter: `users` (Figure 8–39).

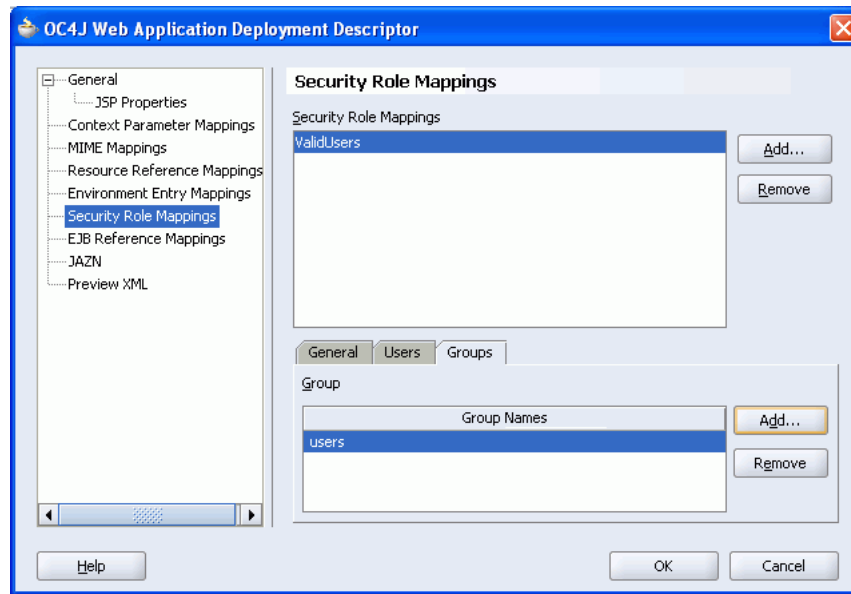
This is the *default* identity store role which maintains a list of every valid user: Singh, Cho, Harvey, JtaAdmin, and oc4jadmin. For details, see [Appendix A, "How to Set Up the Tutorial Identity Store"](#).

**Figure 8–39** Deployment Descriptor - Mapping Group of Users to the J2EE Security Role

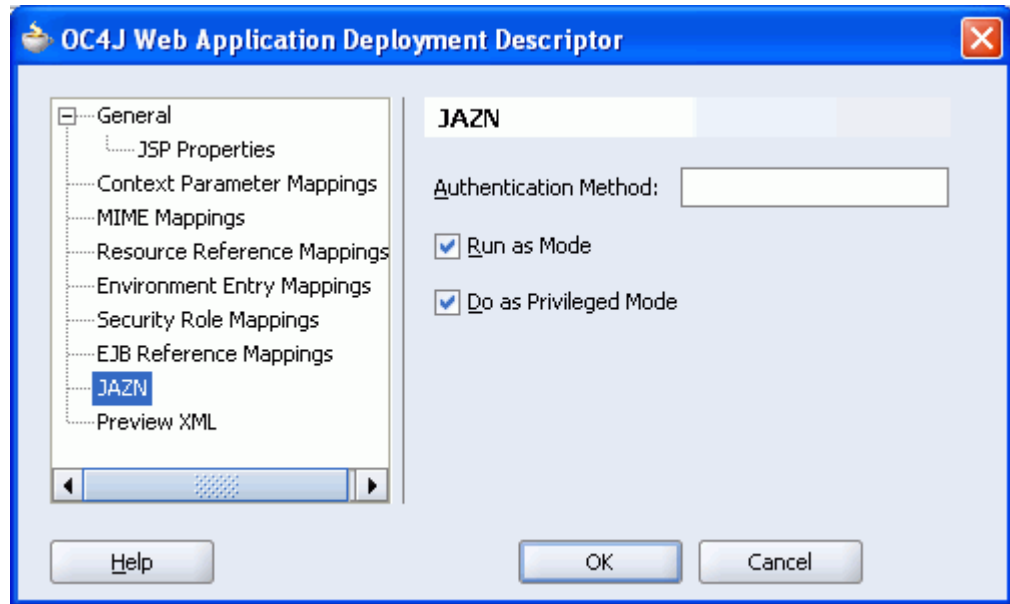
- g. Click **OK**.

In this step you mapped the J2EE security role `ValidUsers` to the identity store role `users` (Figure 8–40).



**Figure 8–40 Deployment Descriptor - J2EE Security Role Mapped to the users Role**

11. Choose **JAZN** in the panel on the left, and select **Run as Mode** and **Do as Privileged Mode** as shown in [Figure 8–41](#).

**Figure 8–41 Deployment Descriptor - JAZN Settings**

12. Click **OK** to save changes to the OC4J deployment descriptor.
13. Click the **Save All** icon in the JDeveloper toolbar.

If you take a look at the source code for the configuration file `orion-web.xml`, you should see `security-role-mapping` entries and a `jazn-web-app` entry as follows:

```
<security-role-mapping name="ValidUsers" impliesAll="false">
  <group name="users"></group>
</security-role-mapping>
```

```
<jazn-web-app runas-mode="true" doasprivileged-mode="true"/>
```

This completes the OC4J Web application deployment descriptor configuration. Now you're ready to run the application and see the new security features in action.

## Step 6: Demonstrating the Security Features

In this final step you'll run the tutorial application and take a look at the security features that you've introduced. Here's a quick summary of what you're going to do:

- Display the welcome page (Welcome.jspx) and test that the login link displays the login page (Login.jspx).
- Use the Login page to enter valid user credentials and check that the correct page gets displayed (Welcome.jspx). Logged in users shouldn't see a login link on the Welcome page—instead, they should see a logout link and, providing they have the necessary page permissions, buttons that navigate to MyPage, MyContent, and MyWeather.
- Login with different user credentials (Singh, Cho, Harvey, and King) and see which page actions each user is allowed to perform.
- Try accessing a secure page directly, without prior authorization, and check that you're redirected to the Login page.
- Enter invalid user credentials and check that the correct error page gets displayed.

Let's run the tutorial application and see these security features in action.

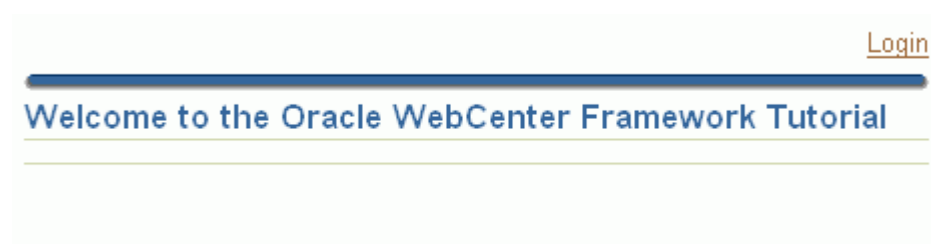
### Login as User Singh

First, let's login as user Singh. This user was assigned *view* privileges on MyPage, MyWeather, and MyContent.

1. In the Applications Navigator, right-click **Welcome.jspx**, and choose **Run**.

In your Browser, you should see a Welcome page with a Login link as shown in [Figure 8-42](#). Earlier on, you configured this link to display the login page Login.jspx.

**Figure 8-42** Welcome Page - Public View



2. Click **Login**.

You should be directed to Login.jspx which contains an entry form for user credentials as shown in [Figure 8-43](#).

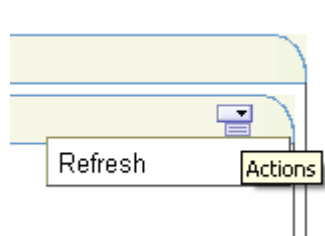
3. Enter login credentials for the user Singh. Both name and password are case-sensitive, so take care when typing them in. For Name enter Singh, and for Password enter welcome.

**Figure 8–43 Login Page - Login Credentials for User Singh**
**4. Click Login.**

If authentication is successful, you should see `Welcome.jspx` with a Logout link at the top of the page, as shown in [Figure 8–44](#). Remember, you defined `Welcome.jspx` as the *success URL* when you configured the Login link on the Welcome page.

**Figure 8–44 Welcome Page - Secured View****5. As user Singh, you should also see several command buttons. Click the button that says **Go To MyPage**.**

As user Singh you have privileges to view this page but you should not be able to personalize or customize anything on the page. To verify this, take a look at the *Actions* menu for MyJavaPortlet.

**6. Click the **Actions** icon in the portlet banner to reveal the available options. As you are logged in as Singh (with view privileges) only the Refresh option should be available, as shown in [Figure 8–45](#).****Figure 8–45 Portlet Actions Available to User Singh****7. Click Logout.**

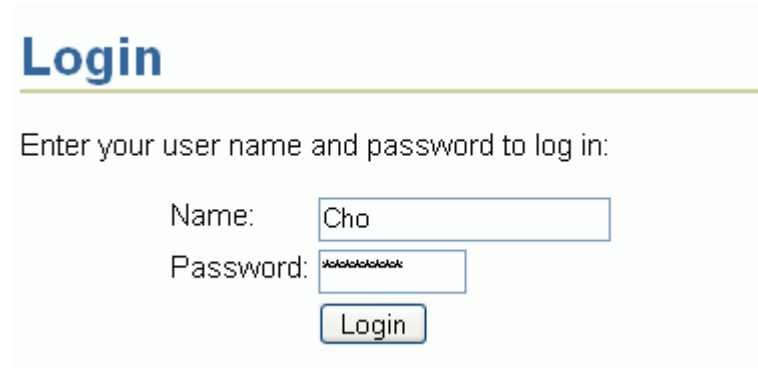
You should get directed back to the Welcome page.

## Login as User Cho

Next, you'll login as user Cho. This user was assigned the page-personalizer role with both view and personalize privileges on MyPage, MyWeather, and MyContent.

1. Display the Welcome page, and click **Login**.
2. Enter login credentials for the user Cho remembering that both fields are case-sensitive. For Name enter Cho, and for Password enter welcome (Figure 8-46).

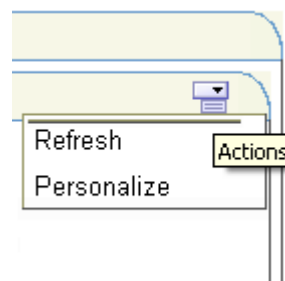
**Figure 8-46 Login Page - Login Credentials for User Cho**



The screenshot shows a login page with the title "Login" in a large blue font. Below the title, there is a prompt: "Enter your user name and password to log in:". There are two input fields: "Name:" with the text "Cho" entered, and "Password:" with masked characters "\*\*\*\*\*". Below the password field is a "Login" button.

3. Click **Login**.  
If authentication is successful, you should see the Welcome page with a Logout link at the top of the page and all three page navigation buttons.
4. Click **Go To MyPage**.  
As user Cho you have privileges to both view and personalize portlets but you should not be able to customize the page or portlets in any way. To verify this, take a look at the *Actions* menu for MyJavaPortlet.
5. Click the **Actions** icon in the portlet banner to reveal the available options. As you are logged in as Cho you should see two options: Refresh and Personalize, as shown in Figure 8-47.

**Figure 8-47 Portlet Actions Available to User Cho**



6. Click **Personalize**.
7. Change the Portlet Title, for example, enter: Cho's Java Portlet  
This title is only displayed when Cho is the logged on user. No other user will see this title.
8. Click **OK** to see the personalizations for user Cho.

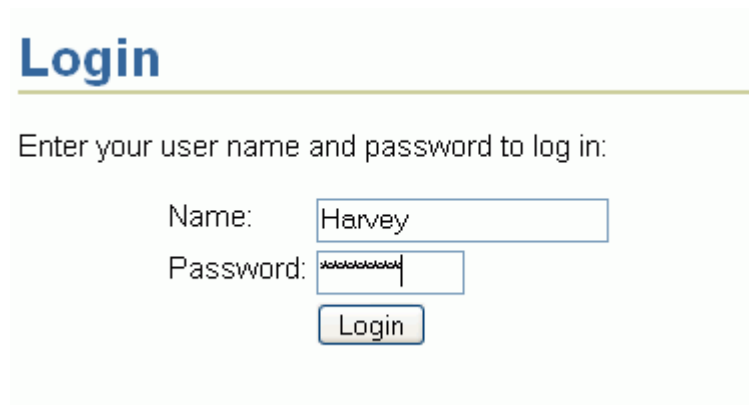
- Click **Logout**.

## Login as User Harvey

Now login as user Harvey. This user was assigned the page-customizer role with view, personalize, and customize privileges on MyPage, MyWeather, and MyContent.

- Display the Welcome page, and click **Login**.
- Enter login credentials for the user Harvey remembering that both fields are case-sensitive. For Name, enter Harvey and for Password enter welcome (Figure 8-48).

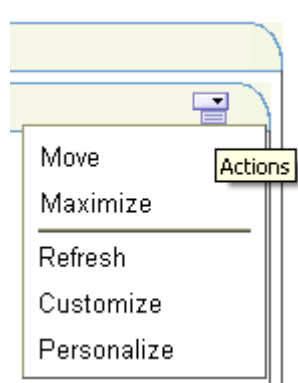
**Figure 8-48 Login Page - Login Credentials for User Harvey**



The screenshot shows a web page titled "Login" with a green header. Below the title, it says "Enter your user name and password to log in:". There are two input fields: "Name:" with the text "Harvey" and "Password:" with masked characters. A "Login" button is positioned below the password field.

- Click **Login**.  
If authentication is successful, you should see Welcome.jspx with a Logout link at the top of the page and several page navigation buttons as before.
- Click **Go To MyPage**.  
As user Harvey you have privileges to view, personalize portlets, and customize this page. To verify this, take a look at the *Actions* menu for MyJavaPortlet.
- Click the **Actions** icon in the portlet banner to reveal the available options. As you are logged in as Harvey you should see five options: Move, Maximize, Refresh, Customize, and Personalize, as shown in Figure 8-49.

**Figure 8-49 Portlet Actions Available to User Harvey**



You can, as user Harvey, customize or move any content on this page and when you do so you'll modify the page for *all* users. This differs from personalizations which are only applicable to the user who is making the changes. Notice that as user Harvey, you do not see the portlet title personalizations made earlier by user Cho.

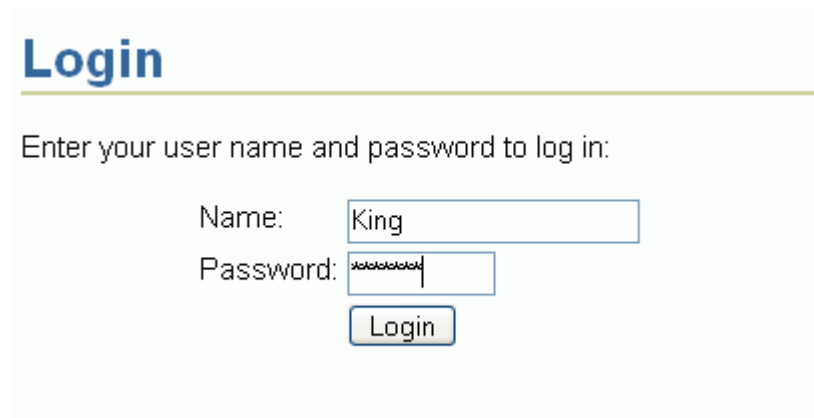
6. Click **Logout**.

## Login as User King

Finally, let's login as user King. This user was assigned the restricted-user role and may access public content only. This user may not view any of the secured pages (MyPage, MyWeather, or MyContent).

1. Display the Welcome page, and click **Login**.
2. Enter login credentials for the user Harvey remembering that both fields are case-sensitive. For Name, enter King, and for Password enter welcome (Figure 8-50).

**Figure 8-50 Login Page - Login Credentials for User King**



The screenshot shows a login page with the title "Login" in a large blue font. Below the title is a horizontal line. Underneath the line, the text "Enter your user name and password to log in:" is displayed. There are two input fields: "Name:" with the text "King" entered, and "Password:" with the text "welcome" entered. Below the password field is a "Login" button.

3. Click **Login**.  
If authentication is successful, you should see Welcome.jspx but you won't see any page navigation buttons as user King is not authorized to view MyPage, MyWeather, or MyContent.
4. Click **Logout**.

## Try to Access a Secured Page Directly

Earlier, you defined a security constraint that enforces login authentication for secured pages. So, if an unauthenticated user tries to access a secured page he should be redirected to the login page for authentication. Let's test this now.

1. In the Applications Navigator, right-click **MyPage.jspx**, and choose **Run**.  
When the Browser opens to display the page the target URL looks something like this:  
`http://123.4.56.789:8988/MySample-ViewController-context-root/faces/MyPage.jspx`  
Because this is a secured page, you should be presented with the login form.
2. Enter valid login credentials such as Harvey/welcome, and click **Login**.

As user Harvey is authorized to view MyPage, you should see MyPage in your browser.

3. Click **Logout**.
4. Try to access MyPage again. In your browser, enter the same URL used in Step 1.
5. Now login as user King who is *not* authorized to view MyPage. Enter `King/welcome`, and click **Login**.

This time, you should see an "Unauthorized" message.

## Enter Invalid Credentials

As part of the tutorial application's login configuration you chose a login error page. Try entering some invalid user credentials to see what happens. You should be directed back to an empty login page.

1. In the Applications Navigator, right-click **Welcome.aspx**, and choose **Run**.
2. When the Welcome page appears in a browser window, click **Login**.
3. Enter some invalid user credentials or just leave both fields blank and see what happens when you click **Login**.

This time you won't see the Welcome page. As an unauthenticated user, you'll be directed back to a blank login form.

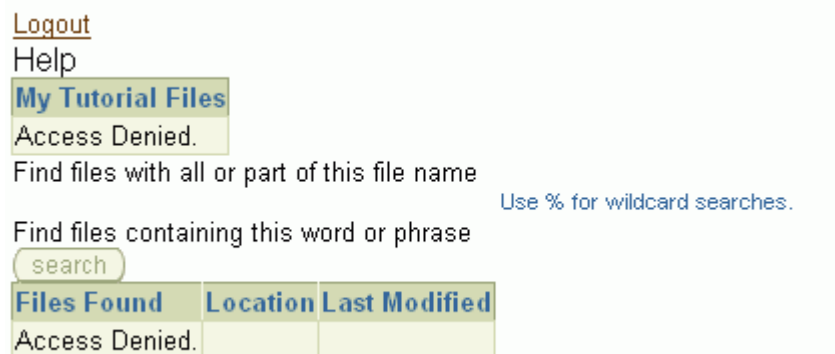
## Step 7: Authorizing Access to Your Data Controls

When you apply access restrictions to a page you automatically restrict access to any data controls on the page. Let's take a look at the data controls you placed on `MyContent.aspx`.

1. In the Applications Navigator, right-click **MyContent.aspx**, and choose **Run**.
2. Enter valid login credentials such as `Harvey/welcome`, and click **Login**.

`MyContent.aspx` should be displayed in your browser but this time you won't have access to file system content (Figure 8-51).

**Figure 8-51** File System Data Controls - Access Denied



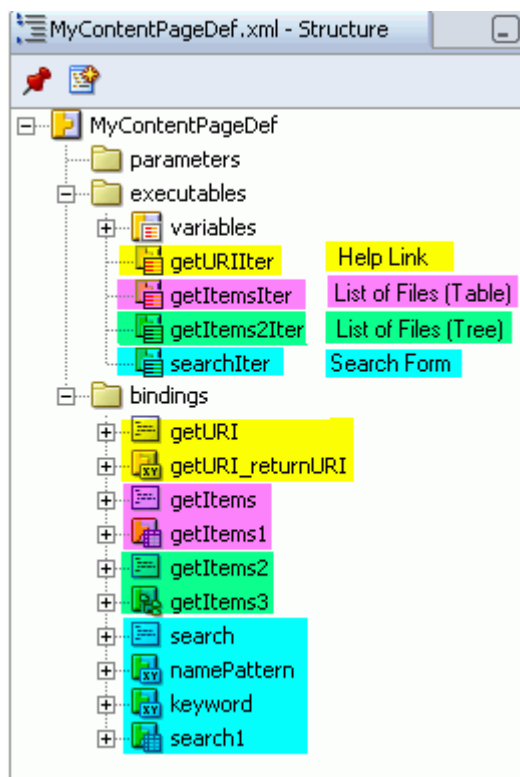
To authorize access, you must grant permissions through the data control's executables and bindings. Let's do that now.

3. Right-click **MyContent.jspx** in the Applications Navigator, and select **Go To Page Definition**.
4. In the Structure window, expand **executables** and **bindings** (Figure 8–52).

To make data control content accessible you must grant permissions through the control's executable (methodIterator) and bindings (methodAction and attributeValues). So, if we want to see the help link on MyContent.jspx we'll need to edit authorization settings for `getURIIter` (methodIterator), `getURI` (methodAction), and `getURI_returnURI` (attributeValues).

Similarly, if we want to see the table of files, we'll need to edit authorization settings for `getItemsIter` (methodIterator), `getItems` (methodAction), and `getItems1` (attributeValues). And, we'd need to do the same for the tree and search form data controls too.

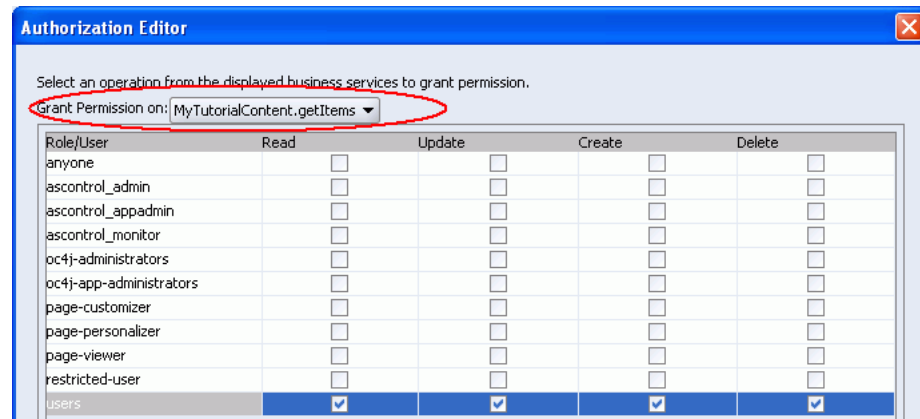
**Figure 8–52 Data Control Executables and Bindings in MyContentPageDef.xml**



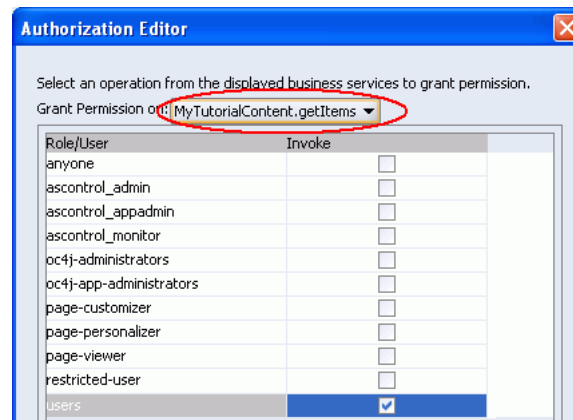
Let's work through all the steps for the *table data control* to see how this is done.

5. First, edit authorizations for the table data control executable `getItemsIter`:
  - a. Under executables, right-click **getItemsIter**, and select **Edit Authorization**.
  - b. Let's grant full permissions to any logged-in user. To do this, check all the boxes for the *users* role as shown in Figure 8–53.



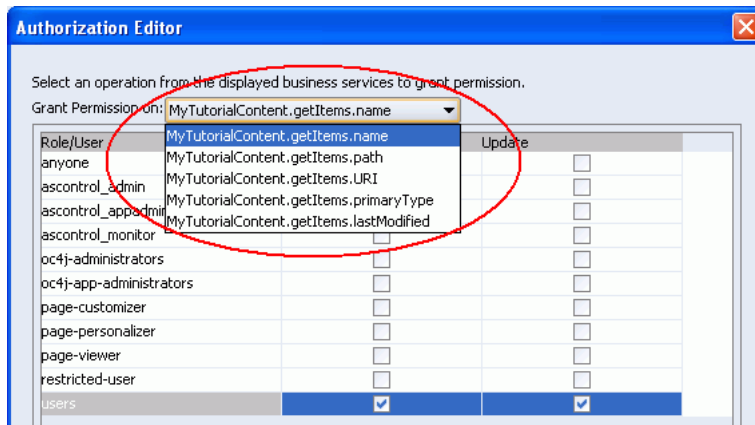
**Figure 8–53 Authorization Editor for Data Control Executables**

- c. Click OK.
6. Now edit authorizations for the methodAction binding of table data control `getItems`:
    - a. Under bindings, right-click `getItems`, and select **Edit Authorization**.
    - b. To grant permission to any logged-in user, check the boxes for the `users` role as shown in [Figure 8–54](#).

**Figure 8–54 Authorization Editor for Data Control methodAction Bindings**

- c. Click OK.
7. And finally, edit authorizations for *each* attributeValues binding of table data control `getItems1`:
    - a. Right-click `getItems1`, and select **Edit Authorization**.
    - b. To grant full permissions to the first attribute **`MyTutorialContent.getItems.name`**, check all the boxes for the `users` role as shown in [Figure 8–55](#).

**Figure 8–55 Authorization Editor for Data Control attributeValues Bindings**



- c. Repeat for every attribute listed in the Grant Permissions box. For example, expand the Grant Permissions box, choose MyTutorialContent.getItems.path, check all the boxes, and repeat.
- d. Click OK.
8. Before running the page again, stop the embedded OC4J Server. Choose **Run, Terminate, Embedded OC4J Server** from the main menu.
9. Run MyContent.jspx again, and login with valid credentials such as Harvey/welcome.

This time you should see files listed in the table data control as shown in Figure 8–56.

**Figure 8–56 Access Authorized to Table Data Control**



If you want, you can enable access to the other data controls on MyContent.jspx by repeating steps 5 through 7.

Congratulations! You've completed the lesson and secured your first WebCenter application. In the final lesson you'll learn about deployment and how to use Enterprise Manager (a browser-based tool that enables administrators to deploy, configure, and manage WebCenter applications).

---

---

## Deploying Your WebCenter Application

In this lesson you will deploy your tutorial application to the WebCenter Preconfigured OC4J. Deploying an application involves packaging up all the required files in a standard J2EE format and directory structure—a WAR file or an EAR file.

All the packaging and deployment instructions for a WebCenter application are configured through a *deployment profile*. A deployment profile is a configuration file that names the pages, portlets, customizations, and metadata comprising the application, the type and name of the archive file to be created, dependency information, platform-specific instructions, and more.

There is a special deployment profile, specifically for WebCenter applications—the *WebCenter Application Deployment Profile*. This profile handles content that is unique to WebCenter applications such as portlets, content repository data controls, customizable component, and so on.

You can deploy WebCenter applications directly from Oracle JDeveloper to a standalone OC4J instance providing that the standalone OC4J and Oracle JDeveloper are on the same computer or have access to a common network drive. You'll learn more about single-step deployment in this lesson. You'll also learn how to manage WebCenter applications through the Application Server Control Console.

### Introduction

You'll deploy the tutorial application by completing the following steps:

- [Step 1: Creating a WebCenter Application Deployment Profile](#)
- [Step 2: Deploying Directly to the Preconfigured OC4J](#)
- [Step 3: Migrating Your Security Policy](#)
- [Step 4: Running Your Deployed Application](#)
- [Step 5: Using Application Server Control Console to Manage WebCenter Applications](#)

### Prerequisites

Before you deploy your WebCenter application:

1. Check that:
  - A connection exists between the WebCenter Preconfigured OC4J and Oracle JDeveloper.
  - The WebCenter Preconfigured OC4J is running.

For more detail, see [Step 2: Setting Up Your Connection \(Chapter 3\)](#).

2. Migrate the tutorial users/roles to the WebCenter Preconfigured OC4J—the deployment target. Run the JAZN Migration Tool in *realm mode* to merge the users and roles on the target OC4J:

- a. Before running the JAZN Migration Tool, update your class path to contain references to: `JDEVHOME\j2ee\home\jazn.jar` and `JDEVHOME\BC4J\lib\adfshare.jar`

For example, enter the following in a command prompt:

```
set
CLASSPATH=JDEVHOME\j2ee\home\jazn.jar;JDEVHOME\BC4J\lib\adfshare.jar
```

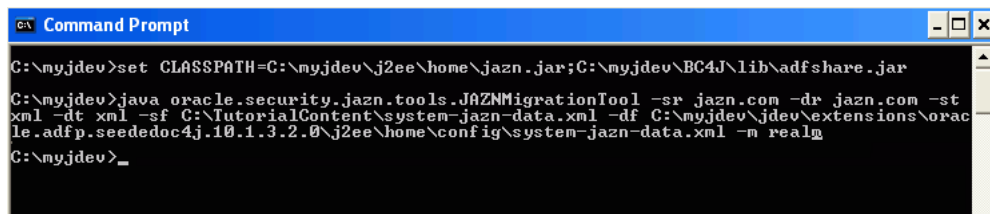
Where *JDEVHOME* points to your JDeveloper installation, for example `C:\myjdev`.

- b. Next, go to *JDEVHOME*, and open a command prompt.
- c. Run the JAZN Migration Tool, as follows:

```
java oracle.security.jazn.tools.JAZNMigrationTool -sr
jazn.com -dr jazn.com -st xml -dt xml -sf
WEBCENTERSAMPLE\TutorialContent\system-jazn-data.xml -df
JDEVHOME\jdev\extensions\oracle.adfp.seededoc4j.10.1.3.2.0
\j2ee\home\config\system-jazn-data.xml -m realm
```

Where *WEBCENTERSAMPLE* points to where you installed the sample tutorial files, for example: `C:`, and *JDEVHOME* points to your JDeveloper installation, for example: `C:\myjdev`. See [Figure 9-1](#).

**Figure 9-1** Running the JAZN Migration Tool



```

C:\myjdev>set CLASSPATH=C:\myjdev\j2ee\home\jazn.jar;C:\myjdev\BC4J\lib\adfshare.jar
C:\myjdev>java oracle.security.jazn.tools.JAZNMigrationTool -sr jazn.com -dr jazn.com -st
xml -dt xml -sf C:\TutorialContent\system-jazn-data.xml -df C:\myjdev\jdev\extensions\orac
le.adfp.seededoc4j.10.1.3.2.0\j2ee\home\config\system-jazn-data.xml -m realm
C:\myjdev>_

```

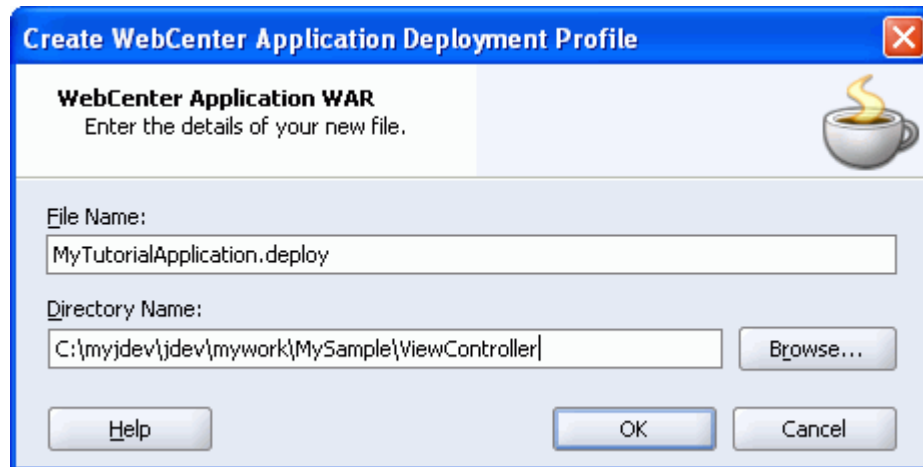
The preconfigured OC4J's `system-jazn-data.xml` file will now contain the users and roles required by your application.

Let's deploy your application.

## Step 1: Creating a WebCenter Application Deployment Profile

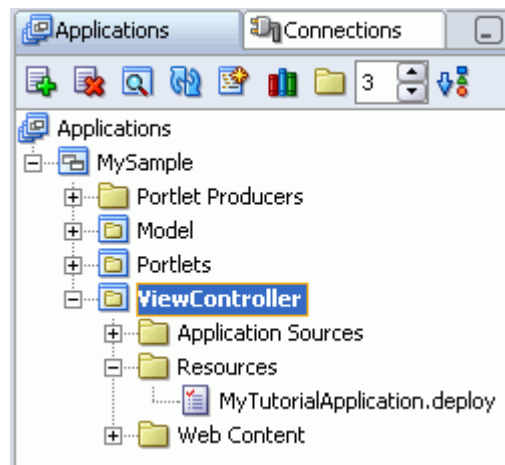
In this step you'll create a deployment profile for the tutorial application called `MyTutorialApplication.deploy`, and set some deployment options.

1. Right-click **ViewController** in the Applications Navigator, and select **New**.
2. Expand the General Category, and choose **Deployment Profiles**.
3. Select **WebCenter Application WAR**, and click **OK**.
4. Enter a name for the deployment profile ([Figure 9-2](#)). For File Name, enter: `MyTutorialApplication`.

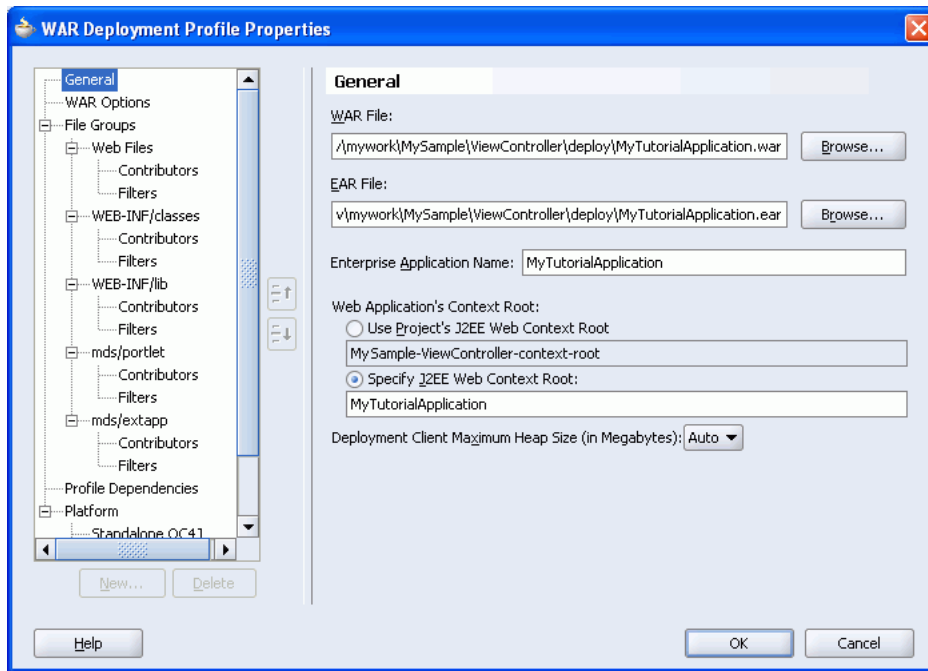
**Figure 9–2 Create WebCenter Application Deployment Profile**

5. Click **OK**.

You should see the new deployment profile displayed under **ViewController**, **Resources** (Figure 9–3).

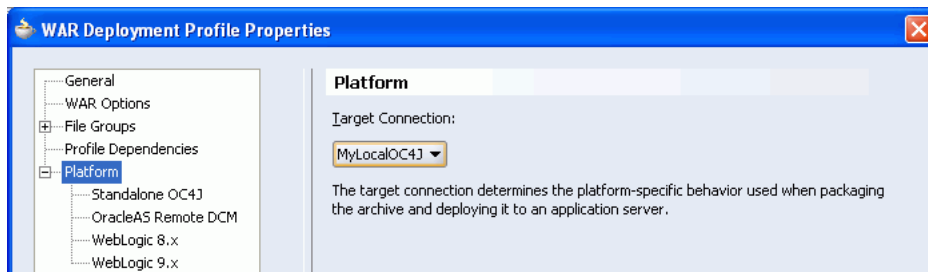
**Figure 9–3 New Deployment Profile - MyTutorialApplication.deploy**

6. To configure the deployment profile, right-click **MyTutorialApplication.deploy** in the Applications Navigator, and select **Properties**.
7. In the WAR Deployment Profile Properties window (Figure 9–4), enter a suitable context root for the application URL. Choose **Specify J2EE Web Context Root**, and enter: `MyTutorialApplication`

**Figure 9–4 WAR Deployment Profile Properties**

8. Select **Platform** from the list on the left, and then choose the connection named **MyLocalOC4J** from the Target Connection dropdown list (Figure 9–5).

You defined this connection earlier on, in [Chapter 3, "Building and Testing Your First Portlet"](#).

**Figure 9–5 WAR Deployment Profile Properties - Target Connection**

You don't need to set any more deployment options on this page. If your application required special libraries, you would select them here.

9. Click **OK**.
10. Click **Save All** in the JDeveloper tool bar.

During development, you can deploy WebCenter applications in a *single step* directly from JDeveloper to a standalone OC4J. You'll see how this works in the next step.

You can deploy your WebCenter applications in a production environment using Application Server Control Console. In the process, you package your WebCenter application in a generic EAR or WAR file. After that, you run the Predeployment tool against this file to remap the WebCenter application's external dependencies, for example, portlet producer end points and the MDS location. The Predeployment tool generates a targeted EAR file that is ready to deploy on the remote system.

Additionally, some tasks related to security, content integration, and external applications may need to be performed as part of the deployment process.

For more information on WebCenter application deployment, see *Oracle WebCenter Framework Developer's Guide*.

## Step 2: Deploying Directly to the Preconfigured OC4J

In this step you'll use the connection details you defined earlier to deploy the tutorial application directly to your preconfigured OC4J. Let's do that now.

1. Right-click **MyTutorialApplication.deploy**, and choose **Deploy to MyLocalOC4J**.

A dialog asks you to define a suitable location for the MDS repository in your file system (Figure 9–6).

2. For MDS Path, enter: `C:\TutorialContent\mds`

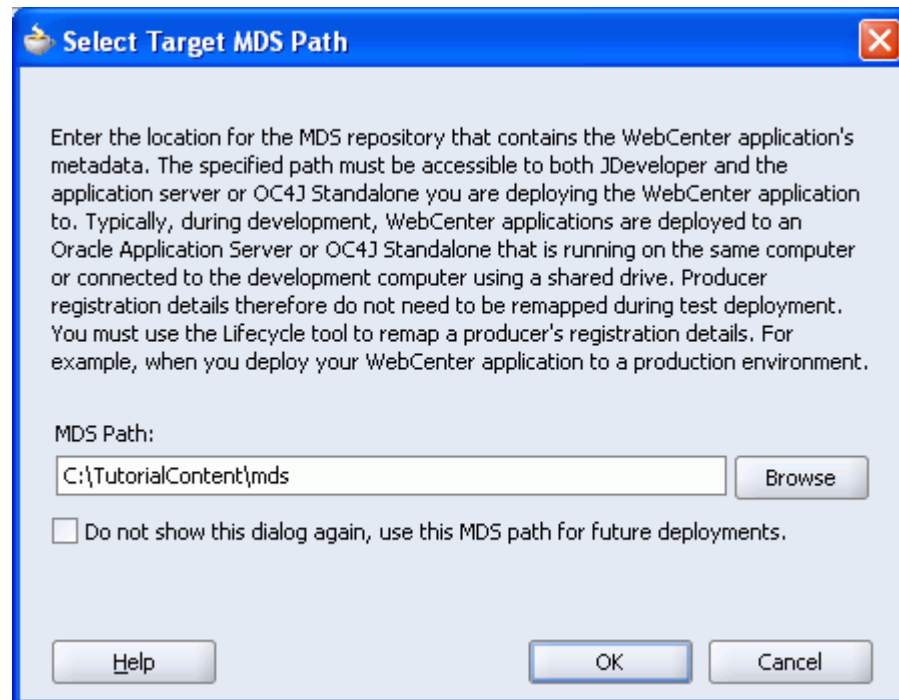
You can choose a different location on your local file system if you wish. If the directory you specify does not exist, JDeveloper will create it for you.

---

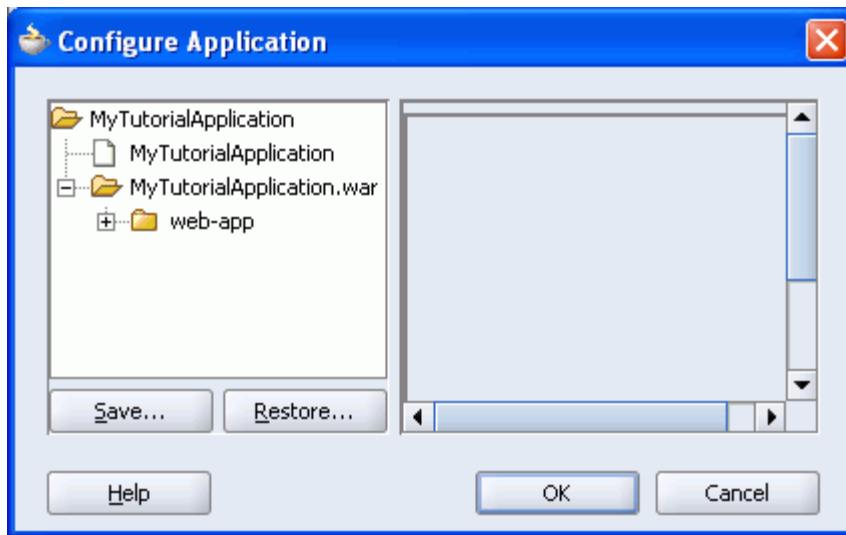
**Caution:** Do **not** create the MDS repository for your deployment inside the working directory for your application (in this case `JDEVHOME\jdev\mywork\MySample`). Keeping separate MDS directories for development and deployment will not only avoid confusion but will prevent content in your application's source MDS repository from being overwritten.

---

Figure 9–6 Target MDS Path



3. Click **OK**.
4. Wait for the Configure Application window to be displayed (Figure 9–7), and then click **OK**.

**Figure 9–7 The Configure Application Window**

WAR and EAR files are generated and deployed to the preconfigured OC4J. The EAR file contain a WAR file and a number of configuration (.xml) files. WAR files contain all the files and libraries used by the application.

At the end of deployment you should see `-- Deployment finished --` in the Deployment-Log window. If you get this message you should be able to access the deployed application but you won't be able to log-in. You still need to migrate your application's security policy (stored in `app-jazn-data.xml`) to the preconfigured OC4J's `system-jazn-data.xml` file. Let's do that now.

## Step 3: Migrating Your Security Policy

In this step you'll use the JAZN Migration Tool to merge your application's security policy (stored in `app-jazn-data.xml`) into the preconfigured OC4J's `system-jazn-data.xml` file.

---

**Note:** You'll find more about the JAZN Migration Tool in the *Oracle WebCenter Framework Developer's Guide*

---

In this tutorial, the location of these XML files are as follows:

### Source `app-jazn-data.xml`

```
JDEVHOME\jdev\extensions\oracle.adfp.seededoc4j.10.1.3.2.0\j2ee\
home\applications\MyTutorialApplication\adf\META-INF\app-jazn-da
ta.xml
```

### Destination `system-jazn-data.xml`

```
JDEVHOME\jdev\extensions\oracle.adfp.seededoc4j.10.1.3.2.0\j2ee\
home\config\system-jazn-data.xml
```

Complete the following steps:

1. Before running the JAZN Migration Tool, update your class path to contain references to these .JAR files:
  - `JDEVHOME\j2ee\home\jazn.jar`
  - `JDEVHOME\BC4J\lib\adfshare.jar`



For example, enter the following in a command prompt:

```
set
CLASSPATH=JDEVHOME\j2ee\home\jazn.jar;JDEVHOME\BC4J\lib\adfshare.jar
```

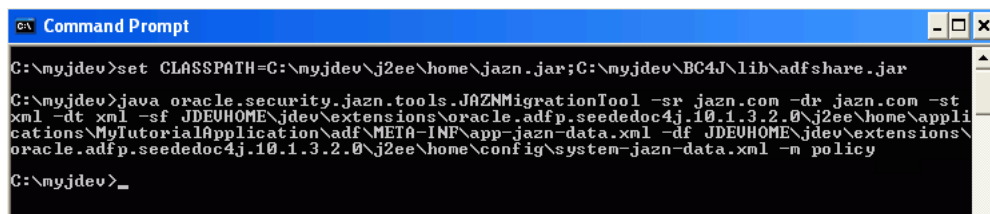
Where *JDEVHOME* points to your JDeveloper installation, for example *C:\myjdev*.

2. Next, go to *JDEVHOME*, and open a command prompt.
3. Run the JAZN Migration Tool, as follows:

```
java oracle.security.jazn.tools.JAZNMigrationTool -sr
jazn.com -dr jazn.com -st xml -dt xml -sf
JDEVHOME\jdev\extensions\oracle.adfp.seededoc4j.10.1.3.2.0\j2
ee\home\applications\MyTutorialApplication\adf\META-INF\app-j
azn-data.xml -df
JDEVHOME\jdev\extensions\oracle.adfp.seededoc4j.10.1.3.2.0\j2
ee\home\config\system-jazn-data.xml -m policy
```

Where *JDEVHOME* points to your JDeveloper installation, for example *C:\myjdev*. For example, see [Figure 9–8](#).

**Figure 9–8** Running the JAZN Migration Tool



The preconfigured OC4J's *system-jazn-data.xml* file will now contain your application's security information. To make this information available, you'll need to restart the WebCenter preconfigured OC4J.

4. To restart the WebCenter preconfigured OC4J from JDeveloper:
  - a. Click the **Stop WebCenter Preconfigured OC4J** icon at the far right of the JDeveloper tool bar.
  - b. Click the **Start WebCenter Preconfigured OC4J** icon.

Wait for the message stating that the OC4J instance has initialized.

Now, let's try to display one of the pages of the deployed application.

## Step 4: Running Your Deployed Application

1. Open a browser window and navigate to the tutorial's Welcome page. You'll need to enter a URL that uses the context root defined in the deployment profile (*MyTutorialApplication*), followed by */faces/*, and the page name itself. The URL format you require is:

```
http://<host>:<port>/<context-root>/faces/<page-name>
```

For example:

```
http://localhost:6688/MyTutorialApplication/faces/Welcome.jsp
x
```

This tutorial assumes that `localhost` can be used in a URL to refer to the local computer on which your preconfigured OC4J installation resides. (If this is not the case, replace `localhost` with your computer's IP address.)

2. Log in as user `Harvey`, and enter the password `welcome`.
3. Click each page navigation button to check that your application works as expected.

Congratulations! You've just deployed your first WebCenter application.

---

**Note:** The WebCenter Preconfigured OC4J is only suitable for test deployments such as this tutorial. Normally, you would not deploy WebCenter applications in the same OC4J instance that is used for the portlet producers. To find out how to deploy to an OC4J in Oracle Application Server or to a standalone OC4J instance, refer to the *Oracle WebCenter Framework Developer's Guide*.

---

## Step 5: Using Application Server Control Console to Manage WebCenter Applications

In this step you'll learn how to use the Oracle Enterprise Manager 10g Application Server Control Console to deploy, redeploy, and undeploy a WebCenter application. You'll also learn how to monitor portlet performance from the console.

1. To access Application Server Control Console for your preconfigured OC4J, navigate to the following URL:

`http://<host>:<port>/em`

For example: `http://localhost:6688/em`

This tutorial assumes that `localhost` can be used in a URL to refer to the local computer on which your preconfigured OC4J installation resides. (If this is not the case, replace `localhost` with your computer's IP address.)

For more information, select **Help, WebCenter Preconfigured OC4J Readme** from the main JDeveloper menu.

2. Log in as user `oc4jadmin`.
3. Enter the default password: `welcome`

The home page for your preconfigured OC4J is displayed.

4. Select the **Applications** tab, and then the name of your application. The application name is derived from the `.EAR` filename. In this tutorial the `.EAR` file was named after the deployment profile — **MyTutorialApplication**.
5. When you click **MyTutorialApplication**, you should see the application's home page, similar to that shown in [Figure 9-9](#).

Figure 9–9 WebCenter Application Home Page in Enterprise Manager

ORACLE Enterprise Manager 10g  
Application Server Control [Setup](#) [Logs](#) [Help](#) [Logout](#)

OC4J: home >  
Application: MyTutorialApplication  
Page Refreshed Dec 18, 2006 7:26:24 PM GMT

[Home](#) [Web Services](#) [Performance](#) [Administration](#)

**General**

[Stop](#) [Restart](#) [Redeploy](#) [Undeploy](#)

Status **Up**  
Start Time **Dec 18, 2006 7:17:06 PM GMT**  
Path **/C:/myjdev/jdev/extensions/  
oracle.adfp.see.doc4j.10.1.3.2.0/  
j2ee/home/applications/  
MyTutorialApplication.ear**  
Parent Application [default](#)

**Related Links**  
[Producers and Portlets](#)

**Modules**

Name ^	Module Type
<a href="#">MyTutorialApplication</a>	Web Module

[Home](#) [Web Services](#) [Performance](#) [Administration](#)

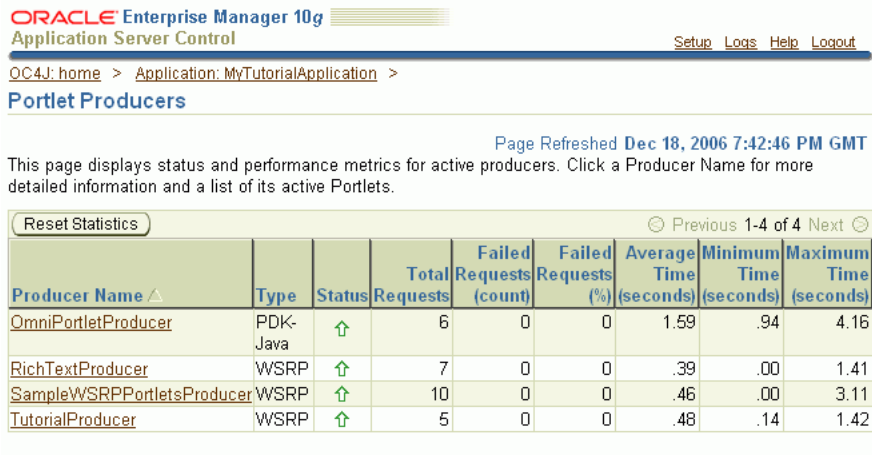
From here you can use the buttons to redeploy and undeploy the application. You can also stop and restart the application from here.

Under **Related Links** you should see a **Producers and Portlets** link. The information available from here is specific to WebCenter applications and it appears only when producer/portlet metrics are available. Statistics become available when a page containing portlets is accessed for the first time. If you did not display a page containing portlets during [Step 4: Running Your Deployed Application](#), do so now (open MyPage or MyWeather), and then refresh the Application Server Control Console.

6. Click **Producers and Portlets** to browse the metrics available for your tutorial application.

You should see the Portlet Producers page (similar to that shown in [Figure 9–10](#)). From this page, you can monitor the status and performance of producers and portlets used by your WebCenter application.

**Figure 9–10 Portlet Producers Screen in Enterprise Manager**



To find out more about deploying, configuring, and monitoring WebCenter applications from the Application Server Control Console, refer to the *Oracle WebCenter Framework Developer's Guide*.

## Summary

In this tutorial you built a simple, secure WebCenter application. You tested and deployed it, then used the Application Server Control Console to monitor its portlet producers.

Now you're more familiar with Oracle JDeveloper and Oracle WebCenter Framework functionality, you can start building your own application. For further assistance, refer to the *Oracle WebCenter Framework Developer's Guide*.

# Part III

---

## Appendixes

Part III contains the following appendix:

- [Appendix A, "How to Set Up the Tutorial Identity Store"](#)



---



---

## How to Set Up the Tutorial Identity Store

This Appendix describes how to set up an identity store—a requirement for [Chapter 8, "Providing Security"](#).

Oracle ADF Security authenticates users against a given resource provider. In this tutorial, we make use of the lightweight XML resource provider `system-jazn-data.xml` supplied with the embedded OC4J. Well-suited to small scale applications like this tutorial, this resource provider is located at:

```
JDEVHOME\jdev\system\oracle.j2ee.10.1.3.xx.xx\embedded-oc4j\conf
ig
```

---



---

**Note:** The system directory is created when you open JDeveloper for the first time.

---



---

For your convenience, we supply a sample `system-jazn-data.xml` file containing all the user data required to complete this tutorial (see [Chapter 2 Downloading Sample Tutorial Files](#) and [Copying the Sample system-jazn-data.xml File](#)). The following table outlines the users/roles our sample file provides:

Role Name	Users	Description
page-viewer	Singh	This user may view secured pages.
page-personalizer	Cho	This user may personalize portlets on a secured page.
page-customizer	Harvey	This user may customize secured pages.
restricted-user	King	This user may not view secured pages.
users	Singh, Cho, King, Harvey, JtaAdmin, oc4jadmin	The users role maintains a list of every valid user.

Only follow the instructions in this Appendix if you would like to enter these tutorial users/roles from scratch for yourself. Maybe you want to learn more about the process, or perhaps you are already building secure applications with JDeveloper and you do not want to overwrite the users, roles, and policies that you've added.

To set up the identity store, completing the steps in the following sections:

- [Creating Users](#)
- [Creating Roles and Assigning User Members](#)
- [Making Tutorial Users and Roles Available to JDeveloper's Authorization Editor](#)

## A.1 Creating Users

In this step, you'll add four users named Singh, Cho, Harvey, and King to the embedded OC4J's `system-jazn-data.xml` file.

1. From the **Tools** menu, choose **Embedded OC4J Server Preferences**.

If the information message *Embedded Server Currently Running* displays, click **No**, and then shut down the embedded OC4J Server (choose **Run, Terminate - Embedded OC4J Server** from the main menu).

2. Under the **Global** branch, expand **Authentication (JAZN)**, **Realms**, and then **jazn.com**.

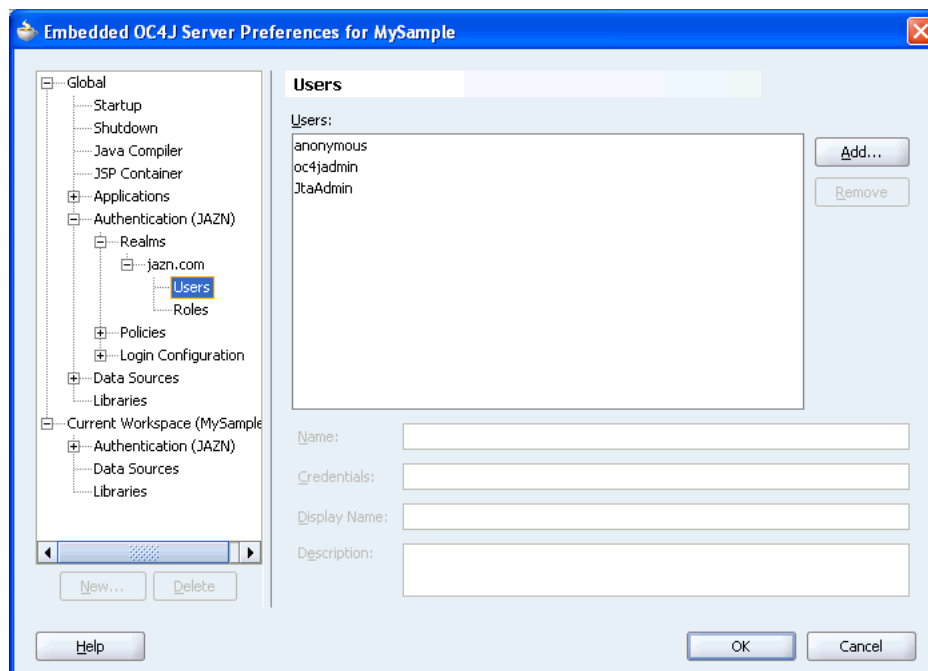
*jazn.com* is the default security realm for the tutorial application.

Don't select the branch called **Authentication (JAZN)** under the **Current Workspace** node. This branch lets you define user data at the application-level but it would not be used by the tutorial application — WebCenter applications only make use of data defined under the global realm.

3. Select **Users**.

You should see three predefined users for the default global security realm, *jazn.com* as shown in [Figure A-1](#).

**Figure A-1** Default Users for Global Security Realm *jazn.com*



The three default users are:

- **anonymous**, a default guest/anonymous user
- **oc4jadmin**, an OC4J administrator
- **JtaAdmin**, another user for recovering propagated OC4J transactions

Do not remove any of these users or some administrative functions will not work.

4. Create a new user named Singh:



- a. Click **Add**.
  - b. For Name, enter Singh.
  - c. In the Credentials field, enter the password welcome.
  - d. Click **OK**. Singh should appear in the Users list.
  - e. For Description, enter `This User may view pages`.
5. Now repeat Step 4. Create *three* more users named Cho, Harvey, and King. Use the credentials and descriptions shown in the following table:

User Name	Credentials	Display Name	Description
Singh	welcome	Singh	This user may view secured pages.
Cho	welcome	Cho	This user may personalize portlets on a secured page.
Harvey	welcome	Harvey	This user may customize secured pages.
King	welcome	Harvey	This user may not view secured pages.

All four new users should appear in the Users list as shown in [Figure A-2](#).

**Figure A-2** New Users

The screenshot shows a web interface for managing users. At the top, there is a title "Users". Below it, a list of users is displayed in a scrollable area. The users listed are: anonymous, oc4jadmin, JtaAdmin, Singh, Cho, Harvey, and King. The "King" user is currently selected and highlighted in blue. To the right of the list are two buttons: "Add..." and "Remove". Below the list, there is a form for editing the selected user. The form fields are: Name (King), Credentials (masked with asterisks), Display Name (King), and Description (This user may not view secured pages.).

6. Click **OK** to save the user definitions in the embedded OC4J's `system-jazn-data.xml`.

## A.2 Creating Roles and Assigning User Members

In this step, you'll add four roles named page-viewer, page-personalizer, page-customizer, and restricted-user to the embedded OC4J's `system-jazn-data.xml` file.

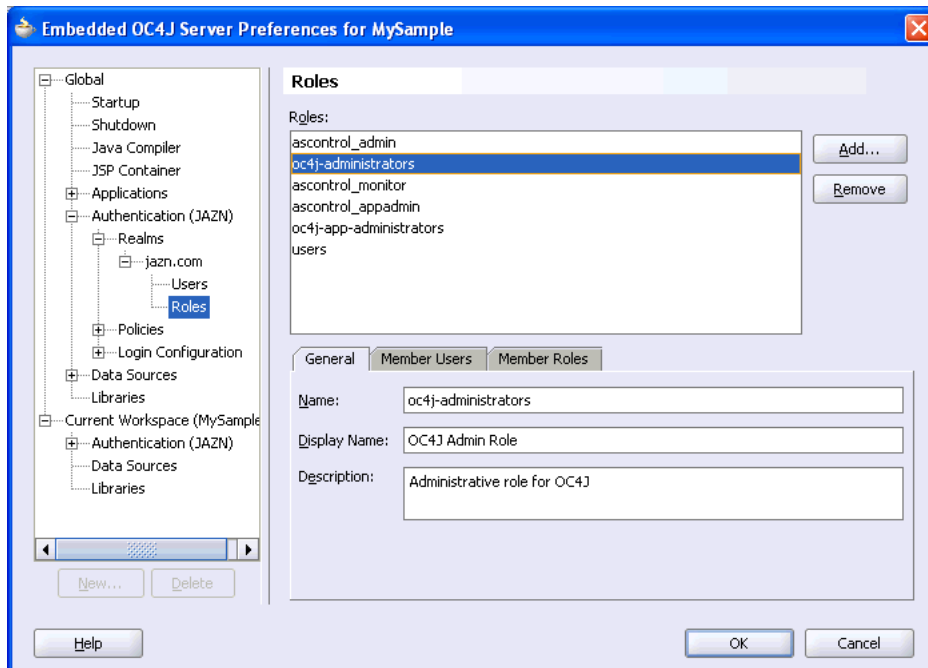
1. From the **Tools** menu, choose **Embedded OC4J Server Preferences**.
2. Expand **Authentication (JAZN), Realms, and jazn.com**.
3. Select **Roles**.

You'll see several predefined roles for the default global security realm `jazn.com`, as shown in [Figure A-3](#):

- **oc4j-administrators**, an OC4J administrator role
- **oc4j-app-administrators**, an OC4J application administrator role
- **users**, a generic group to map all users in the system.
- **ascontrol\_admin**, an Enterprise Manager Application Server Control administrator role
- **ascontrol\_appadmin**, an Enterprise Manager application administrator role
- **ascontrol\_monitor**, an Enterprise Manager monitoring role

Do not remove any of these roles, or some administrative functions will not work. For more information, see *Oracle Application Server Administrator's Guide*.

**Figure A-3** Default Roles for the Global Security Realm `jazn.com`



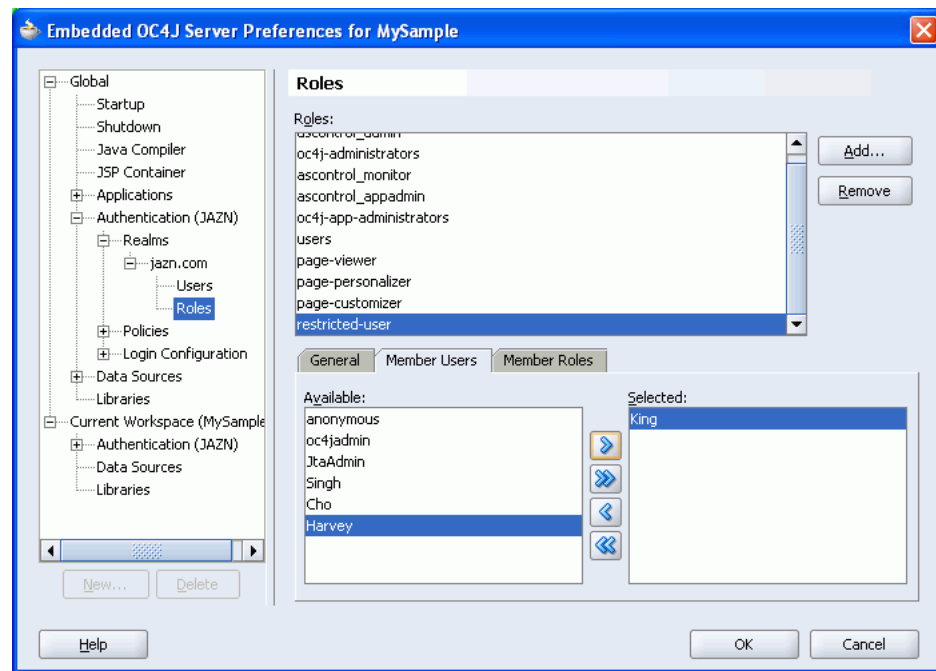
4. Create a new role named page-viewer, and assign user Singh to this role.
  - a. Click **Add**.
  - b. Enter the Name `page-viewer`, and click **OK**.
  - c. Click the **Member Users** tab, and move **Singh** to the list on the right.

5. Now repeat Step 4. Add three more roles and assign a member user to each role as outlined in this table:

Role	Member Users
page-viewer	Singh
page-personalizer	Cho
page-customizer	Harvey
restricted-user	King

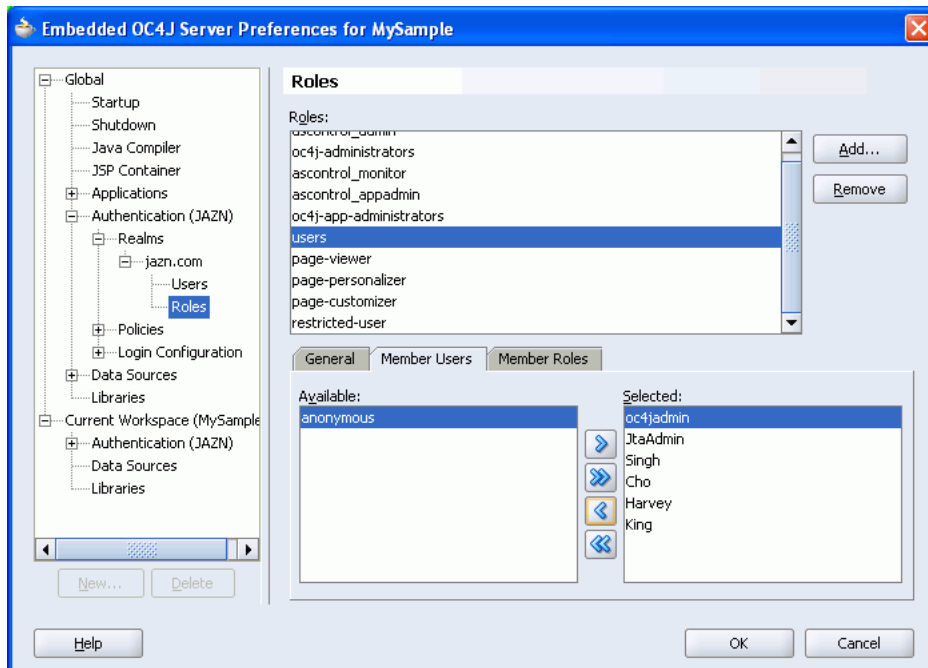
- a. Add roles named page-personalizer, page-customizer, and restricted-user.
- b. Assign member Cho to the page-personalizer role, member Harvey to the page-customizer role, and member King to the restricted-user role as shown in [Figure A-4](#).

**Figure A-4** Member Users Assigned to New Roles



6. Place all the users (except anonymous) into the *users* role:
  - a. Select the **users** role.
  - b. Click the **Member Users** tab, and move users (Singh, Cho, Harvey, King, JtaAdmin, and oc4jadmin) to the list on the right as shown in [Figure A-5](#).

**Figure A-5 Assigning Members to the Users Role**



The users role maintains a list of every valid user. In [Chapter 8, "Providing Security"](#), you map this role to a J2EE security role called `ValidUsers` (for details, see [Step 2: Configuring ADF Security Settings](#)).

7. Click **OK** to save the role definitions to the embedded OC4J's `system-jazn-data.xml` file.

In the next step, you'll make these users/roles available through the Authorization Editor in Oracle JDeveloper. You assign page permissions through this editor in [Chapter 8 Step 4: Securing Pages](#).

### A.3 Making Tutorial Users and Roles Available to JDeveloper's Authorization Editor

In this step you'll copy the tutorial users/roles to JDeveloper' home directory so they are available to JDeveloper design-time dialogs.

1. Before making any modifications for the purposes of this tutorial, back up the `system-jazn-data.xml` file located at `JDEVHOME\j2ee\home\config`.
2. Copy the `system-jazn-data.xml` file from the embedded OC4J directory `JDEVHOME\jdev\system\oracle.j2ee.10.1.3.xx.xx\embedded-oc4j\config` to `JDEVHOME\j2ee\home\config`.

---

---

**Note:** If you already have a populated `system-jazn-data.xml` file at this location you must *merge* the files rather than overwriting the original. Run the JAZN Migration Tool in *realm mode* to merge the users and roles:

1. First, set the CLASSPATH to:  
`JDEVHOME\j2ee\home\jazn.jar;JDEVHOME\BC4J\lib\adfshare.jar`
2. Run the JAZN Migration Tool with the following syntax:  
`java oracle.security.jazn.tools.JAZNMigrationTool -sr jazn.com -dr jazn.com -st xml -dt xml -sf JDEVHOME\jdev\system\oracle.j2ee.10.1.3.xx.xx\embedded-oc4j\config\system-jazn-data.xml -df JDEVHOME\j2ee\home\config\system-jazn-data.xml -m realm`

Where *JDEVHOME* points to your JDeveloper installation, for example `C:\myjdev`, and *10.1.3.xx.xx* refers to the version number. For more information, see the *Oracle WebCenter Framework Developer's Guide*.

---

---



## A

---

ADF Security  
  see Oracle ADF Security, 8-1  
ADF, see Application Development Framework, 1-6  
adfAuthentication servlet, 8-6  
  granting access to, 8-8  
ADFContext library, 8-14  
advancedSearch method, 7-4  
anyone role, 8-23  
app-jazn-data.xml  
  about, 8-24  
  migrating security information, 9-6  
Application Development Framework (ADF), 1-6  
authentication  
  defining users and roles, A-1  
  enabling, 8-5  
  form-based, 8-7  
  testing, 8-30  
Authorization Editor  
  data controls, 8-36  
  pages, 8-23

## B

---

buttons (page navigation)  
  adding, 8-18  
  hiding from unauthorized users, 8-20

## C

---

CacheResults property (content), 7-16  
connection to preconfigured OC4J, 3-10  
Content DB, 1-5  
content publishing, 7-1  
content repository data control, 7-2  
context root, 9-3  
customizable components  
  definition, 4-2  
  editing properties, 4-9  
customize mode  
  enabling, 3-6  
  testing, 3-25

## D

---

data control

  defining, 7-2  
  see also file system data control, 7-2  
deployment, 9-1  
  deploying to preconfigured OC4J, 9-5  
  using Enterprise Manager, 9-8  
deployment descriptor  
  defining, 8-26  
deployment profile  
  WAR file, 3-13  
  WebCenter Application WAR, 9-2  
Do As Privileged mode, 8-6, 8-29

## E

---

EAR deployment, 9-1  
embedded OC4J  
  copying sample users and roles, 2-2  
  defining users and roles, A-1  
  terminating, 3-29  
  testing applications, 3-25  
Enterprise Manager  
  accessing the Application Server Control  
  console, 9-8  
  monitoring producers and portlets, 9-9

## F

---

faces-config.xml, 8-18  
file system data control  
  applying security, 8-35  
  default attributes, 7-3, 7-10, 7-20  
  defining, 7-2  
  methods, 7-4  
  publishing a file, 7-5  
  publishing content in a table, 7-8  
  publishing content in tree format, 7-17  
files  
  publishing a file link, 7-5  
  publishing content in a table, 7-8  
  publishing content in a tree, 7-17  
folders  
  publishing content in a table, 7-8  
  publishing content in a tree, 7-17

## G

---

- getAttributes method, 7-4
- getItems method, 7-4
  - publishing content in a table, 7-8
  - publishing content in tree format, 7-17
- getURL method, 7-4
  - publishing file links, 7-5
- graphics
  - adding to a page, 4-4
  - adding to the Rich Text Portlet, 5-5

## I

---

- identity store
  - assigning page privileges, 8-23
  - creating roles, A-4
  - creating users, A-2
  - setting up system-jazn-data.xml, A-1
  - using tutorial sample, 2-2
- images
  - adding to a page, 4-4
  - adding to the Rich Text Portlet, 5-5

## J

---

- J2EE security roles
  - configuring, 8-8
  - mapping identity store roles, 8-25
- J2EE Web Context Root, 9-3
- JAAS Mode, 8-6
- JavaServer Faces page
  - creating, 3-16
  - running, 3-25
- JAZN Migration Tool
  - merging application security policy, 9-6
  - merging users/roles, 9-2
  - setting class path, 9-2
- JAZN settings, 8-7, 8-29
- jazn.com, 8-7, A-2
- JCR adapters, 1-4
- JCR data controls, 1-4
- JDeveloper
  - see Oracle JDeveloper, 1-6
- JSF navigation rule
  - defining, 8-18
  - JSF Navigation Diagram, 8-19
- JSR 168 Java portlet
  - building, 3-4
  - deploying, 3-13
  - registering, 3-20
  - testing, 3-22

## L

---

- library requirements, 3-18
- lifecycle support, 1-4
- lightweight XML provider, 8-6
- login link
  - defining, 8-12
  - hiding from authenticated users, 8-14

- Login page
  - creating, 8-2
  - error page, 8-7, 8-35
- Login.jspx, 8-2
- logout link
  - hiding from unauthenticated users, 8-17
  - on MyContent.jspx, 8-25
  - on MyPage.jspx, 8-24
  - on MyWeather.jspx, 8-25
  - on Welcome.jspx, 8-16

## M

---

- MDS repository, 9-5
- MyContent.jspx
  - authorizing access to data controls, 8-35
  - authorizing access to the page, 8-25
  - creating, 7-5
- MyPage.jspx
  - authorizing access to, 8-24
  - creating, 3-16
- MyTutorialContent (data control), 7-2
- MyTutorialContent2 (data control), 7-17
- MyWeather.jspx, 6-1
  - authorizing access to, 8-25
  - creating, 6-5
- MyWeatherPageDef.xml, 6-7, 6-11

## N

---

- navigation rule
  - see JSF navigation rule, 8-18
- NDValue, 7-15
- nt file, 7-15
- nt folder, 7-15

## O

---

- OC4J
  - deployment descriptor, 8-26
  - JDeveloper's embedded, 3-25
  - starting the preconfigured, 3-10
- OmniPortlet, 6-1
  - adding to a page, 6-10
  - configuring a Web Service, 6-12
  - description, 1-3
  - portlet parameters, 6-11
  - producer registration, 6-2
  - using, 6-10
- Oracle ADF Security, 8-1
  - enabling, 8-5
  - Security Wizard, 8-5
  - settings, 8-5
- Oracle Content Database, 1-5
- Oracle JDeveloper
  - and ADF, 1-6
  - downloading, 2-1
  - embedded OC4J, 3-25
- Oracle PDK portlet producer, 6-2
- Oracle Technology Network, viii
- Oracle WebCenter Suite, 1-2



orion-web.xml, 8-25  
OTN  
    see Oracle Technology Network, viii

## P

---

page navigation  
    command buttons, 8-18  
    see JSF navigation rule, 8-18  
page navigation buttons  
    adding, 8-18  
    hiding from unauthorized users, 8-20  
page permissions, 8-22  
page variables  
    mapping portlet parameters, 6-16  
    OmniPortlet parameter mappings, 6-11  
    Parameter Form Portlet parameter mappings, 6-7  
panelCustomizable  
    description, 4-2  
    using, 4-3  
Parameter Form Portlet, 6-1  
    adding to a page, 6-5  
    customization, 6-9  
    passing parameters, 6-15  
    portlet parameters, 6-5, 6-7  
    producer registration, 6-2  
parameters  
    portlet parameters, 6-5, 6-10  
    Web Service parameters, 6-12  
personalize mode, 3-6, 8-32  
portlet  
    allowing users to customize, 3-7  
    allowing users to personalize, 3-6  
    building a JSR 168, 3-4  
    deploying to OC4J, 3-10, 3-13  
    exposing as a web service, 3-16  
    files generated for, 3-10  
    registering with WebCenter Framework, 3-20  
    testing, 3-22  
portlet communication, 6-1  
    configuring, 6-15  
    testing, 6-16  
portlet parameters  
    mapping to page variables, 6-8  
    OmniPortlet, 6-11  
    Parameter Form Portlet, 6-5, 6-7  
    passing parameters, 6-15  
portlet producers  
    accessible through Preconfigured OC4J, 3-11  
    monitoring through Enterprise Manager, 9-9  
    Oracle PDK portlets, 6-2  
    WSRP portlets, 6-2  
portlet provider  
    see WAR file, 3-13  
preconfigured OC4J  
    accessing sample portlets, 6-1  
    accessing the Application Server Control  
        console, 9-8  
    connecting to, 3-10  
    deploying tutorial application, 9-5

installing, 3-10  
migrating security policy, 9-2  
readme, 3-11  
starting and stopping, 3-10  
test pages, 5-2

Predeployment Tool, 1-5, 9-4  
producer registration  
    Omniportlet, 6-2  
    Rich Text portlet, 5-2  
    sample WSRP portlets, 6-2

## R

---

RangeSize property (content), 7-16, 7-23  
Rich Text portlet  
    adding to a page, 5-3  
    customizing at run time, 5-5  
    description, 1-4  
    producer registration, 5-2  
roles  
    anyone role, 8-23  
    assigning page privileges, 8-23  
    default roles (embedded OC4J), A-4  
    defining (embedded OC4J), A-4  
    mapping in orion-web.xml, 8-25  
    migrating to preconfigured OC4J, 9-2, A-6  
    see also J2EE security roles, 8-8  
rules (Tree Binding Editor), 7-20  
Run As mode, 8-29

## S

---

sample files  
    copying sample system-jazn-data.xml, 2-2  
    downloading, 2-1  
search method, 7-4  
security  
    ADF Security settings, 8-5  
    authorizing access to data controls, 8-35  
    authorizing access to pages, 8-22  
    creating a login page, 8-2  
    creating a Welcome page, 8-10  
    setting up an identity store, A-1  
showDetailFrame  
    description, 4-2  
    using, 4-6  
system-jazn-data.xml  
    adding users and roles, A-1  
    Authorization Editor, 8-23  
    copying sample data, 2-2  
    defining permissions on data controls, 8-36  
    defining permissions on pages, 8-23  
    making available to Authorization Editor, A-6  
    Oracle ADF Security, 8-6  
    preparing user data, 8-2

## T

---

tag library requirements, 3-18

## U

---

### users

- assigning to roles (embedded OC4J), A-4
- default users (embedded OC4J), A-2
- defining users (embedded OC4J), A-2
- migrating to preconfigured OC4J, 9-2, A-6

## W

---

### WAR file

- creating, 3-13, 9-2
- deployment, 9-1

### Weather Web Service

- configuring, 6-12
- parameters, 6-12

### web application archive

- see WAR file, 3-13

### Web Clipping portlet, 1-4

### Web Service see Weather Web Service, 6-12

### WebCenter application

- context root, 9-3
- creating new, 3-2
- deployment, 9-1
- deployment profile, 9-2
- security, 8-1
- WAR, 1-4, 9-2

### WebCenter Framework, description, 1-3

### WebCenter Services, 1-5

### web.xml, 8-5, 8-9

### Welcome page

- authorizing public access, 8-22
- creating, 8-10
- login link, 8-12
- logout link, 8-17
- page navigation buttons, 8-18

### Welcome.jsp, 8-10

### WSDL

- publishing a portlet as, 3-16

### WSRP portlet producer

- creating, 3-20
- Parameter Form Portlet, 6-2

## X

---

### XML resource provider, 8-5