Oracle® WebCenter Framework

Building a WebCenter Application Step by Step

10*g* (10.1.3.2.0)

B31073-01

June 2007

Oracle WebCenter Framework Building a WebCenter Application Step by Step 10*g* (10.1.3.2.0)

B31073-01

Contributing Authors: Lalithashree Rajesh, Peter Lubbers, Promila Chitkara, Vanessa Wang

Contributors: Barry Hiern, Candace Fender, Oliver Ricordel, Peter Moskovits, Philipp Weckerle, Sue Vickers

Contents

Preface vii

Audience vii
Documentation Accessibility vii
Related Documents viii
Conventions viii

1 Introduction to the WebCenter Suite Example

What is Oracle WebCenter Suite? 1-1
Oracle 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 Life Cycle 1-5
Oracle WebCenter Services 1-5
Oracle JDeveloper 1-6
What Will I Create in this Example? 1-7
How Do I Get Started? 1-9
How Do I Navigate the Completed Demo? 1-9
Running the Demo 1-10
Viewing the Demo as a Customer 1-11
Viewing the Demo as a Technician 1-11
Viewing the Demo as a Manager 1-11

2 Building Your Portlets

Before You Begin 2-1
Creating a Project for Your Portlets 2-1
Creating an Application Server Connection to the Oracle WebCenter Preconfigured OC4J 2-3
Step 1: Building the Product Details Portlet (a JSR 168 Portlet) 2-4
Creating a JSR 168 Portlet 2-5
Creating a Web Service Proxy for Your JSR 168 Portlet 2-7
Adding Portlet Logic to a JSR 168 Portlet 2-10
Deploying a JSR 168 Portlet to an Application Server 2-11
Register the Producer 2-14

Step 2: Building the Service Request Status Portlet (a PDK-Java Portlet) 2-16
Creating a PDK-Java Portlet and Producer 2-16
Adding Portlet Logic to a PDK-Java Portlet 2-21
Deploying a PDK-Java Portlet to an Application Server 2-26
Summary 2-28

3 Setting Up Skins

Step 1: Verifying that the Skins are Registered 3-1
Step 2: Verifying that Your Application is Configured to Use the New Skins 3-2
Summary 3-2

4 Setting Up Your Content Repository

Step 1: Setting Up a Content Directory for the Sample Content 4-1
Step 2: Creating a Content Project 4-1
Step 3: Configuring a JCR Data Control 4-3
Summary 4-6

5 Creating a Public Welcome Page

Step 1: Adding a Rich Text Component 5-1
Step 2: Displaying a File In Place 5-4
Step 3: Adding Global Navigation Links 5-6
Step 4: Making the Welcome Page Public 5-11
Summary 5-13

6 Creating a Login Page

Step 1: Creating a Login Page 6-2
Step 2: Creating a Login Error Page 6-11
Step 3: Adding a Rich Text Portlet to the Login Page 6-14
Step 4: Editing Authorization for the Login Page 6-15
Step 5: Configuring the Application to Use ADF Security and the Login Page 6-16
Step 6: Updating Web.xml to Call the New Login Page 6-18
Step 7: Running the Application 6-20
Summary 6-20

7 Building a Page and Adding Components

Step 1: Creating the MyAcme Page 7-2
Step 2: Registering the OmniPortlet Producer 7-4
Step 3: Adding Customization Components 7-5
Step 4: Wiring the SelectOneChoice to the ADF Table 7-11
Step 5: Adding a JavaServer Faces Drop-Down Component 7-12
Step 6: Adding a Service Request History Portlet 7-13
Step 7: Adding the Service Request Status Portlet 7-14
Step 8: Wiring the Table to the Portlets 7-14
Wire the OmniPortlet to the Table 7-15

Wire the Service Request Status and the Service Request History Portlets to the Service Request Table 7-16
Step 9: Adding a Product Details Portlet 7-16
Step 10: Displaying Folder Content 7-17
Step 11: Adding the Schedule On-Site Services Portlet 7-18
Registering a Web Clipping Producer 7-19
Adding the Web Clipping Portlet 7-19
Selecting the Web Page to Display in the Web Clipping Portlet 7-19
Step 12: Adding a Current Contract Based on the User 7-22
Step 13: Applying Security to the My Acme Page 7-23
Step 14: Applying Security to the Components 7-24
Summary 7-24

8 Building a Dashboard Page

Step 1: Creating the Dashboard Page 8-2
Step 2: Creating the Page Layout 8-4
Step 3: Adding Instances of OmniPortlet to Your Page 8-9
Step 4: Adding a SelectOneChoice Component for the Service Request Volume Portlet 8-10
Step 5: Adding a Search for Customer Contracts 8-11
Step 6: Defining the Most Productive Employees Portlet 8-11
Step 7: Defining the Service Request Volume Portlet 8-12
Step 8: Defining the Most Requested Products Portlet 8-13
Step 9: Defining the Most Active Customers Portlet 8-13
Step 10: Defining the Customer Details Portlet 8-14
Step 11: Wiring the Page Content Together 8-15
Summary 8-16

9 Building a Site Administration Page

Step 1: Creating the Site Administration Page 9-2
Step 2: Adding the Skin Selector to the Page 9-3
Step 3: Enabling Customization of the Login Page 9-7
Step 4: Adding the Site Administration Page as a Subtab of the Management Page 9-9
Step 5: Using the Site Administration Page for Customization 9-10
Changing the Skin for the Application 9-10
Customizing the Login Page 9-11
Summary 9-13

10 Deploying Your Application

Step 1: Creating the Generic EAR file 10-1
Step 2: Creating the Targeted EAR File 10-3
Step 3: Setting Up the Users and Roles 10-5
Step 4: Setting Up the Sample Content 10-6
Step 5: Deploying the Application 10-6
Step 6: Deploying the Security Policy 10-8
Step 7: Accessing the Application 10-9

Summary 10-9

Index

Preface

This manual describes how to build a WebCenter application using Oracle WebCenter Framework. The example used in this manual is based on the ADF application shown in the *Oracle Application Development Framework Developer's Guide*, and shows you how to add portal-like capabilities to an existing Oracle ADF application.

Audience

This manual is intended for the WebCenter application developer who wants to build a WebCenter application, or the application developer who wants to use Oracle Oracle WebCenter Framework to add customization capabilities to their application. Accordingly, this manual assumes that the developer is already familiar with:

- Java
- Oracle JDeveloper
- Oracle Application Development Framework (Oracle ADF)
- Oracle ADF Faces
- *Oracle WebCenter Framework Tutorial*

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, see the following documents:

- *Oracle WebCenter Framework Release Notes*
- *Oracle WebCenter Framework Developer's Guide*

Conventions

The following text conventions are used in this document:

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

1 Introduction to the WebCenter Suite Example

This chapter introduces you to the Oracle WebCenter Suite, its features, and an overview of the scenario used in this book.

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

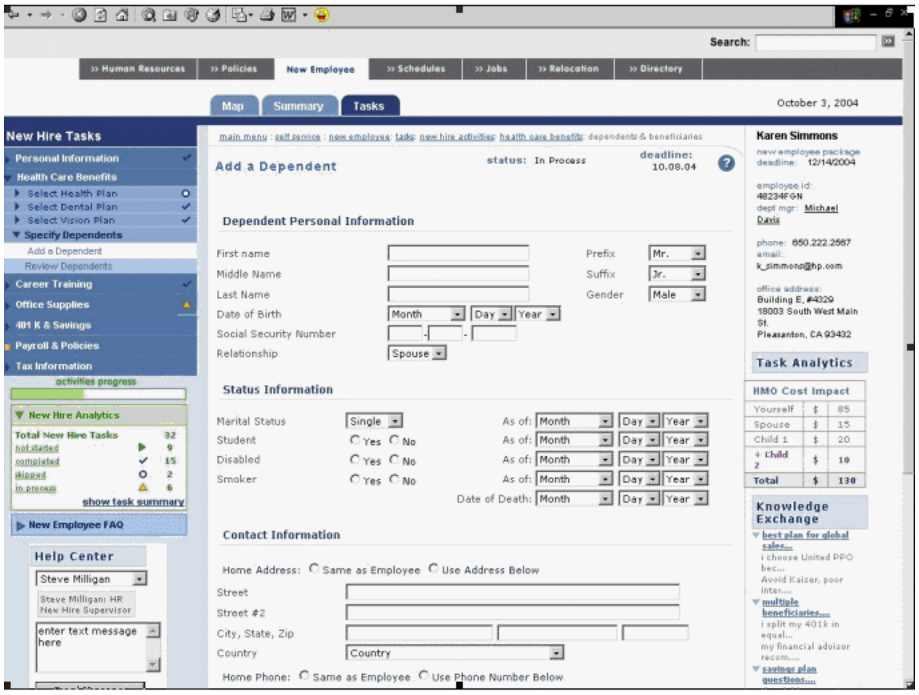
- What is Oracle WebCenter Suite?
- What Will I Create in this Example?
- How Do I Get Started?
- How Do I Navigate the Completed Demo?

After you read this chapter, you'll be ready to start creating your own WebCenter 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



October 3, 2004

New Hire Tasks
- Personal Information ✓
- Health Care Benefits
  - Select Health Plan ○
  - Select Dental Plan ✓
  - Select Vision Plan ✓
  - Specify Dependents
    - Add a Dependent
    - Review Dependents
- Career Training ✓
- Office Supplies ⚠
- 401 K & Savings
- Payroll & Policies
- Tax Information

activities progress

New Hire Analytics

Total New Hire Tasks		32
not started	▶	9
completed	✓	15
skipped	○	2
in process	⚠	6

show task summary

New Employee FAQ

Help Center

Steve Milligan

Steve Milligan: HR New Hire Supervisor

enter text message here

main menu : self service : new employee: tasks: new hire activities: health care benefits: dependents & beneficiaries

Add a Dependent

status: In Process

deadline: 10.08.04

Dependent Personal Information

First name		Prefix	Mr.
Middle Name		Suffix	Jr.
Last Name		Gender	Male
Date of Birth	Month Day Year		
Social Security Number	- -		
Relationship	Spouse		

Status Information

Marital Status	Single	As of:	Month Day Year
Student	○ Yes ○ No	As of:	Month Day Year
Disabled	○ Yes ○ No	As of:	Month Day Year
Smoker	○ Yes ○ No	As of:	Month Day Year
		Date of Death:	Month Day Year

Contact Information

Home Address: ○ Same as Employee ○ Use Address Below

Street	
Street #2	
City, State, Zip	
Country	Country

Home Phone: ○ Same as Employee ○ Use Phone Number Below

Karen Simmons

new employee package deadline: 12/14/2004

employee id: 48234FGN

dept mgr: Michael Davis

phone: 650.222.2567

email: k_simmons@hp.com

office address:
Building E, #4329
18003 South West Main St.
Pleasanton, CA 93432

Task Analytics

HMO Cost Impact		
Yourself	$	85
Spouse	$	15
Child 1	$	20
+ Child 2	$	10
Total	$	130

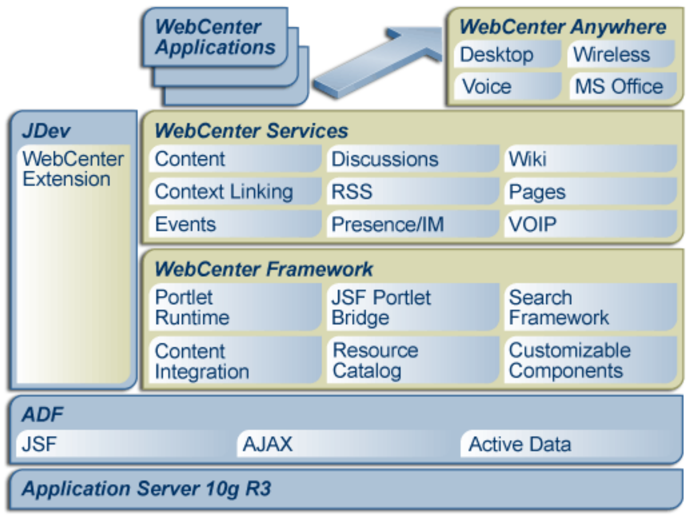
Knowledge Exchange
- best plan for global sales...
  - i choose United PPO bec...
  - Avoid Kaiser, poor inter....
- multiple beneficiaries....
  - i split my 401k in equal...
  - my financial advisor recom....
- savings plan questions....

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

- 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 multistep flow quickly and easily.
- Task and process analytics let users know where they are in the process and how decisions are affecting them. In this case, the Task Analytics on the right show the total cost effect 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 JCR 1.0 Java Content Repository (JSR 170)--if the application programming interface (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 JCR 1.0 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[1]:

Figure 1–2 Oracle WebCenter Suite



Let's examine these building blocks in more detail.

Oracle WebCenter Framework

Oracle WebCenter Framework augments the Java Server Faces (JSF) environment by providing additional integration and run-time customization options. In essence, it integrates capabilities historically included in Oracle Application Server Portal (OracleAS 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 prebuilt portlets are available for use through a preconfigured Oracle Containers for J2EE (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

[1] Some components shown are not available in the initial release of Oracle WebCenter Suite: Presence/IM, Discussions, and Wiki. This chapter provides a description of components relevant to this release of Oracle WebCenter Framework.

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, then so is the user's associated Web Clipping.
- Rich Text portlet: A tool that enables 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 enable 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 or maximizing, hiding or 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 must 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 Oracle 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. This manual contains an illustration of how to create a publicly accessed login page that allows users to enter their credentials and thus gain access to various pages or components within a page. Examples are also provided for how to create login/logout links and how to associate privileges with various roles.

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 Life Cycle

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

- Development framework: Oracle JDeveloper and Oracle ADF provide the tools and framework you must 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 enable 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.

Oracle WebCenter Services

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

- 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 do the following:
  - 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 Information Technology (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.

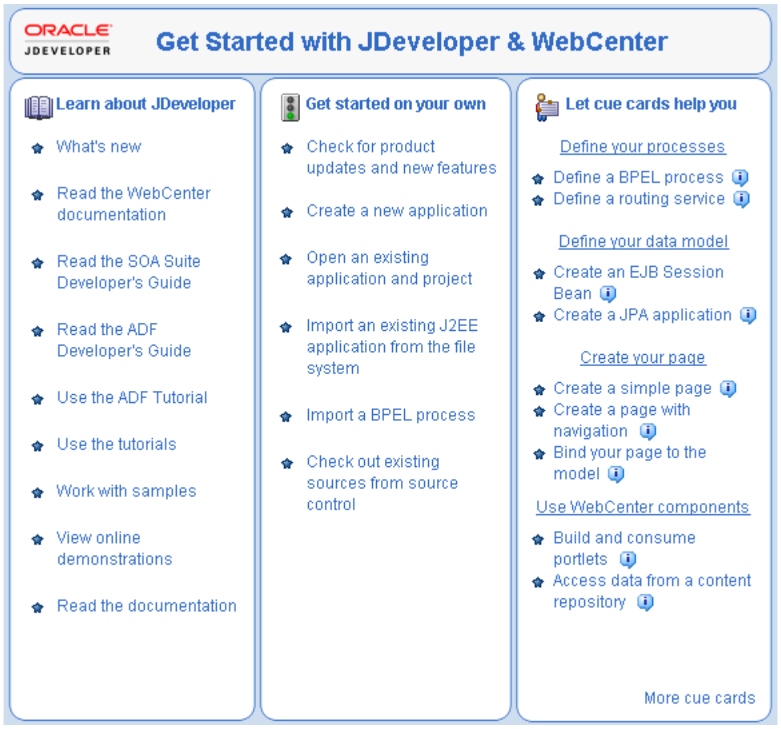
- Oracle Secure Enterprise Search is a crawler-based service that can search a multitude of sources, structured and unstructured, in a variety of file formats, indexed or real-time. With Oracle Secure Enterprise Search, you can reduce the time spent finding relevant documents on your company's information repositories.
- Communication Services, which help you better connect people and facilitate communication. These services include the following:
  - 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 enables 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 Oracle 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 prepackaged 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 guide, 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


ORACLE
JDEVELOPER
Get Started with JDeveloper & WebCenter
Learn about JDeveloper
What's new
Read the WebCenter documentation
Read the SOA Suite Developer's Guide
Read the ADF Developer's Guide
Use the ADF Tutorial
Use the tutorials
Work with samples
View online demonstrations
Read the documentation
Get started on your own
Check for product updates and new features
Create a new application
Open an existing application and project
Import an existing J2EE application from the file system
Import a BPEL process
Check out existing sources from source control
Let cue cards help you
Define your processes
Define a BPEL process
Define a routing service
Define your data model
Create an EJB Session Bean
Create a JPA application
Create your page
Create a simple page
Create a page with navigation
Bind your page to the model
Use WebCenter components
Build and consume portlets
Access data from a content repository
More cue cards

What Will I Create in this Example?

The steps in this manual show you how to add portlets, integrate existing content, and add customization to your application, as well as create interactivity. Such interactivity includes portlet-to-portlet communication and ADF Faces component-to-portlet interactivity. You will also learn how to allow administrators to customize pages at runtime.

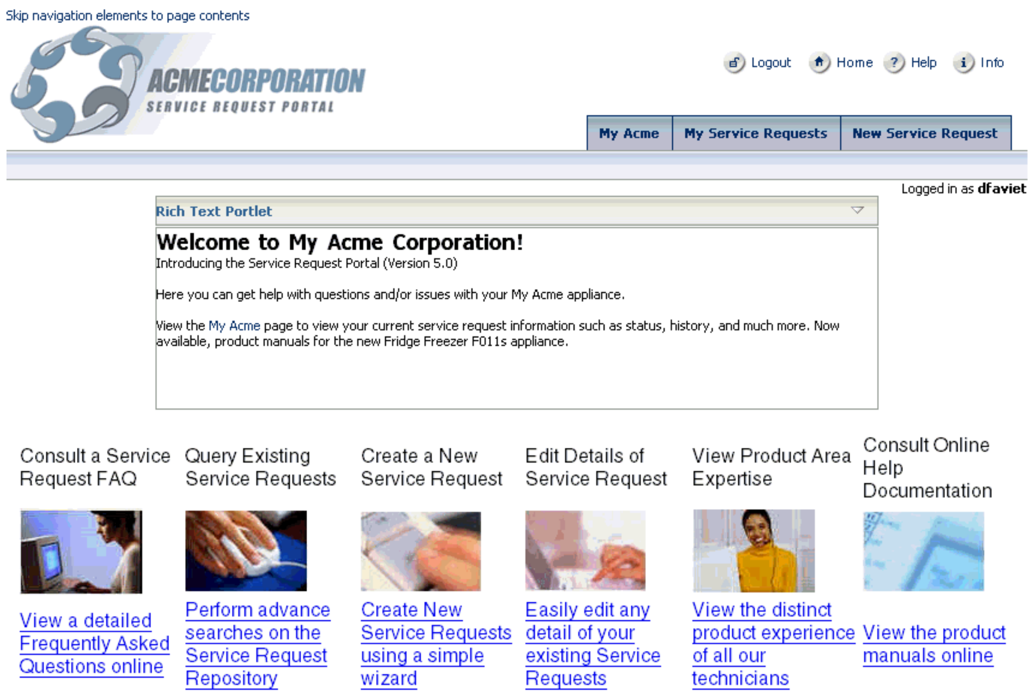
In this manual, you will take an existing ADF application for tracking customer service requests and add portal capabilities to the application without altering the existing application. This service request application enables customers, technicians, and managers view information about service requests all from the same interface. The three scenarios are:

- Customer (a customer of the My Acme Corporation)
- Technician (a technician who handles service requests for the My Acme Corporation)
- Manager (a manager who administers the Web site and runs a team of technicans for the My Acme Corporation)

Customer

The customer logs into the application and can view current announcements, his existing service requests, and details about these requests, as shown in Figure 1–4. He can also view information about the products he has purchased, as well as a list of his current contracts with the company providing the services. He can also submit feedback on existing service requests.

Figure 1–4 Customer View of the Service RequestApplication



Welcome to My Acme Corporation!
Introducing the Service Request Portal (Version 5.0)

Here you can get help with questions and/or issues with your My Acme appliance.

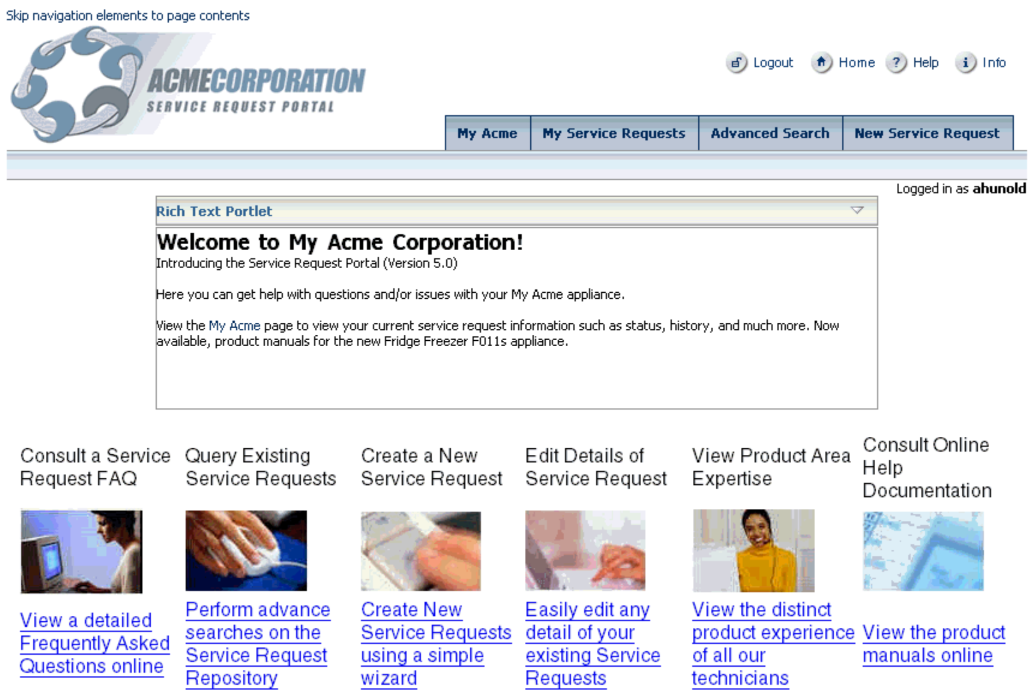
View the My Acme page to view your current service request information such as status, history, and much more. Now available, product manuals for the new Fridge Freezer F011s appliance.

Consult a Service Request FAQ	Query Existing Service Requests	Create a New Service Request	Edit Details of Service Request	View Product Area Expertise	Consult Online Help Documentation
View a detailed Frequently Asked Questions online	Perform advance searches on the Service Request Repository	Create New Service Requests using a simple wizard	Easily edit any detail of your existing Service Requests	View the distinct product experience of all our technicians	View the product manuals online

Technician

The technician logs into the application and views the service requests assigned to him, as shown in Figure 1–5, and can update existing service requests.

Figure 1–5 Technician View of the Service Request Application



Welcome to My Acme Corporation!
Introducing the Service Request Portal (Version 5.0)

Here you can get help with questions and/or issues with your My Acme appliance.

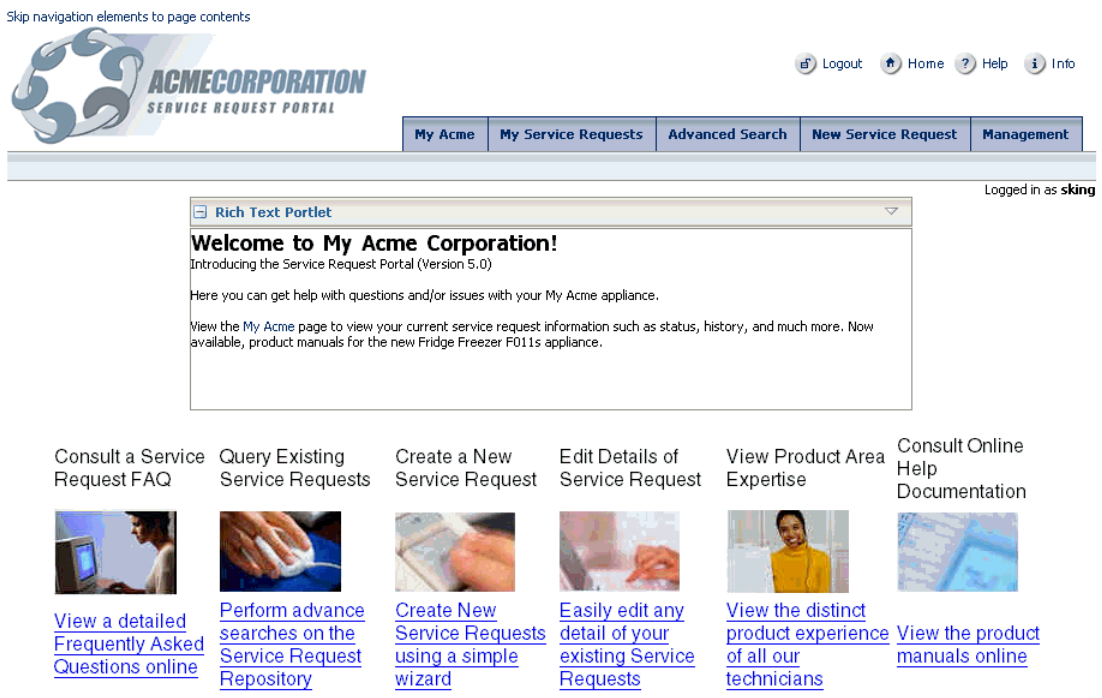
View the My Acme page to view your current service request information such as status, history, and much more. Now available, product manuals for the new Fridge Freezer F011s appliance.

Consult a Service Request FAQ	Query Existing Service Requests	Create a New Service Request	Edit Details of Service Request	View Product Area Expertise	Consult Online Help Documentation
View a detailed Frequently Asked Questions online	Perform advance searches on the Service Request Repository	Create New Service Requests using a simple wizard	Easily edit any detail of your existing Service Requests	View the distinct product experience of all our technicians	View the product manuals online

Manager

The manager logs into the application and, at runtime, can update the announcements that the customer views. He can also modify the page at runtime using content in the content repository. For example, if a new service is now available to customers, the manager can add information about this new service at runtime. The manager can also review the feedback the customer has returned and add his own notes. He can also view site statistics through a dashboard page, which shows the current service request volume, the most active customers, and so on, as well as customize this dashboard page. The manager also has access to general site administration from a single page, where he can change the look and feel by switching skins and customize the login. Figure 1–6 shows the manager's view of the application.

Figure 1–6 Manager View of the Service Request Application



Welcome to My Acme Corporation!
Introducing the Service Request Portal (Version 5.0)

Here you can get help with questions and/or issues with your My Acme appliance.

View the My Acme page to view your current service request information such as status, history, and much more. Now available, product manuals for the new Fridge Freezer F011s appliance.

Consult a Service Request FAQ — View a detailed Frequently Asked Questions online

Query Existing Service Requests — Perform advance searches on the Service Request Repository

Create a New Service Request — Create New Service Requests using a simple wizard

Edit Details of Service Request — Easily edit any detail of your existing Service Requests

View Product Area Expertise — View the distinct product experience of all our technicians

Consult Online Help Documentation — View the product manuals online

How Do I Get Started?

Before you begin, download and set up the sample application and supporting files. The sample files associated with this manual are located on the Oracle WebCenter Suite Documentation page on Oracle Technology Network (OTN) at `http://www.oracle.com/technology/products/webcenter/documentation.html`. Download and extract the file `SRDemo_App_Download.ZIP`, then follow the instructions in the `install.html` file. After you have set up the files, you can view the completed demo, as described in the next section, or continue to Chapter 2, "Building Your Portlets" to start building the application.

How Do I Navigate the Completed Demo?

After you install and set up the sample files, you can view the completed, working version of the demo so you can see what you are going to build in the following chapters. This section describes the demo and how to use the demo at runtime.

Running the Demo

To run the demo:

1. In your browser, go to the following URL:

   `http://localhost:8888/SRDemo/faces/app/SRWelcome.jspx`

   > **Note:** The application uses portlets that already have a hostname for its URL or DB connection. You will need to modify the hostname for the application to be fully functional.

2. Log into the application using the following login information:

   User ID: `sking`

   Password: `welcome`

3. The first page that displays is the welcome page for the Service Request application. The page is visible to the public (unauthenticated) users, and contains a Help and About page. There are two components on the welcome page: a Rich Text portlet and HTML content from a content repository.

   - The Rich Text Portlet provides an easy way for site administrators to provide new announcements, text, and other HTML updates at runtime directly from the running application. Users who are not logged in can only view and refresh the portlet.
   - The images and text for the welcome page is HTML content located in a file-based content repository.

4. Click the **My Acme** tab to set up and view some of the other components in the application.

5. On the My Acme page, click the **Schedule On-Site Service** sub-tab to access the Web Clipping portlet.

6. Click **Customize**. Then, you can follow the steps in Section , "Step 11: Adding the Schedule On-Site Services Portlet" to select the Web page for the portlet.

7. Next, you will need to set up the database connection for the OmniPortlets in the demo, so that you can run them. Click the **SR Information** sub-tab.

8. Next to the Service Request History portlet, click Customize.

9. On the Source tab, click **Edit Connection**, then enter the database connection details. The database connection information should be the same as the information you entered when you installed the sample files.

   > **Note:** Once you edit the connection of one OmniPortlet instance in the demo, all OmniPortlet instances should use the same connection information (for example, all the portlets on the My Dashboard page should now work properly). You can always customize each OmniPortlet if you find they do not use the same connection.

10. Click **Finish**. You should now be able to use the portlets in the application.

Viewing the Demo as a Customer

There are three different types of users built into the application: customer, technician, and manager. To view the application from a customer's perspective, log in using the following information:

- User ID: dfaviet
- Password: welcome

As the customer you can view the status and history of your service requests. Click the MyAcme tab and then select a service request to view more details.

Viewing the Demo as a Technician

To view the application from a customer's perspective, log in using the following information:

- User ID: ahunold
- Password: welcome

As the technician, you can also view customer service request status and histories. Click the My Acme tab and select one of the open service requests to view the latest information about that service request. You can modify that service request on the My Service Requests tab.

You also have access to additional components that a customer is not able to see. On the My Acme page, on the Product Information sub-tab, a new component (JSR 168 portlet) lets you view product details about the specific washers, dryers, and so on.

Finally, you have access to the Schedule On-Site Service sub-tab where an external application enables you to view the next available on site service that can be scheduled.

Viewing the Demo as a Manager

To view the application from a customer's perspective, log in using the following information:

- User ID: sking
- Password: welcome

As the manager, you see all tabs available within the application. Click the Management tab, then click the Dashboard sub-tab to see information about service requests within the application. You can view the following details:

- Search for customer contracts. For example, type "`df%`" and click **Search** to view the contact for the customer `dfaviet`.
- View details of the most active customers. For example, click **Urman** to view the customer details for Jose Manul Urman.
- View the service volume over a specified time period.

The manager is also the Site Administrator. Click the **Management** tab again, then click **Site Administration**. You can modify the skin of the application by selecting one of the available skins.

2 Building Your Portlets

In this chapter, you will learn how to build two different types of portlets: a Java portlet that uses JSR 168 and a PDK-Java portlet that uses Oracle-specific APIs.

This chapter contains the following sections:

- Before You Begin
- Step 1: Building the Product Details Portlet (a JSR 168 Portlet)
- Step 2: Building the Service Request Status Portlet (a PDK-Java Portlet)

Before You Begin

Before you begin the steps in this chapter, ensure that you have completed the steps in the `install.html` file located in the sample files you downloaded. Then, perform the following steps:

- Creating a Project for Your Portlets
- Creating an Application Server Connection to the Oracle WebCenter Preconfigured OC4J

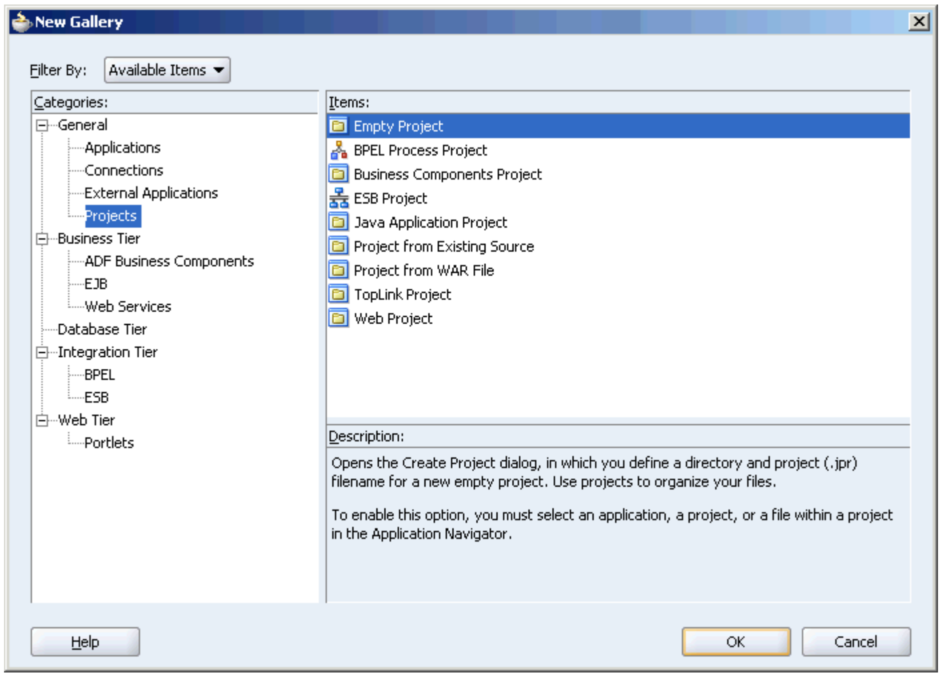
Creating a Project for Your Portlets

To make it easy to find things in your application, you can create projects to contain your portlets.

To create a project for your portlets:

1. In the Applications Navigator, right-click the **SRDemoSample _Starter** application and select **New Project**.
2. In the Items list, shown in Figure 2–1, select **Empty Project** and click **OK.**

Figure 2–1 Create an Empty Project

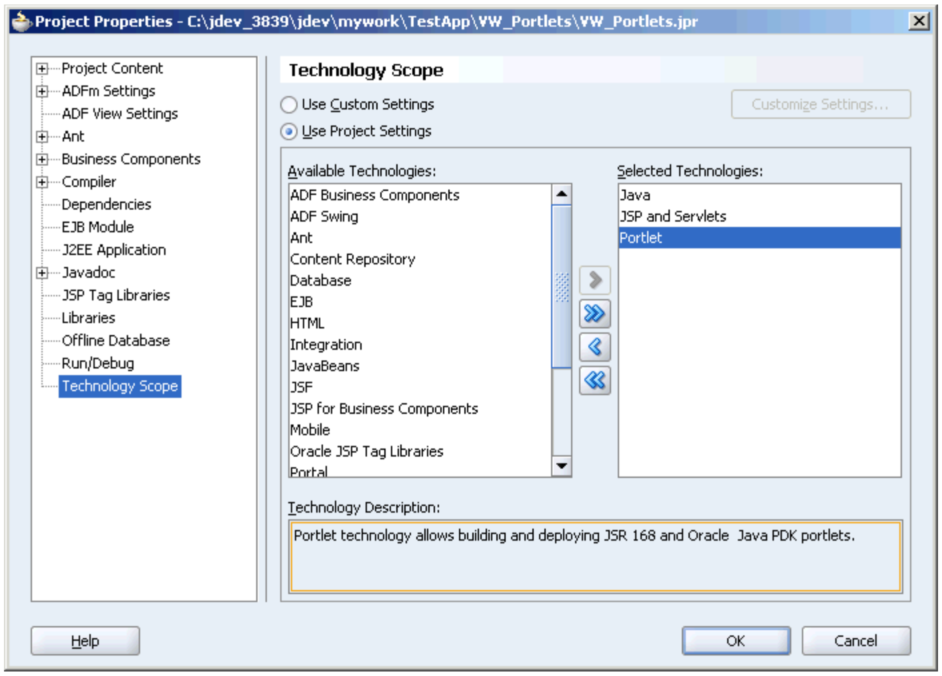


Opens the Create Project dialog, in which you define a directory and project (.jpr) filename for a new empty project. Use projects to organize your files.

To enable this option, you must select an application, a project, or a file within a project in the Application Navigator.

3. In the **Project Name** field, enter `ProdDetailPortlet`.
4. Use the default directory name, then click **OK** to create the project.
5. To view the project properties, right-click the project name (in this case, **ProdDetailPortlet**), then choose **Project Properties** from the pop-up menu.
6. Select the technology scope of the project (that is, the technologies you will be including in the project): Java, JSP and Servlets, and Portlet, as shown in Figure 2–2.

Figure 2–2 Technology Scope of Your New Project



Project Properties - C:\jdev_3839\jdev\mywork\TestApp\VW_Portlets\VW_Portlets.jpr

Technology Scope

Use Custom Settings

[✓ Use Project Settings]

Available Technologies:

ADF Business Components
ADF Swing
Ant
Content Repository
Database
EJB
HTML
Integration
JavaBeans
JSF
JSP for Business Components
Mobile
Oracle JSP Tag Libraries
Portal

Selected Technologies:

Java
JSP and Servlets
Portlet

Technology Description:

Portlet technology allows building and deploying JSR 168 and Oracle Java PDK portlets.

7. Click **OK**.
8. Create a second project by following steps 1 through 7 again, and name the project `SRStatusPortlet`.
9. Save the application.

Creating an Application Server Connection to the Oracle WebCenter Preconfigured OC4J

Before you can deploy portlets to your application serverOracle JDeveloper, you first need to establish a connection to your Oracle WebCenter Preconfigured OC4J, which is embedded in Oracle JDeveloper.

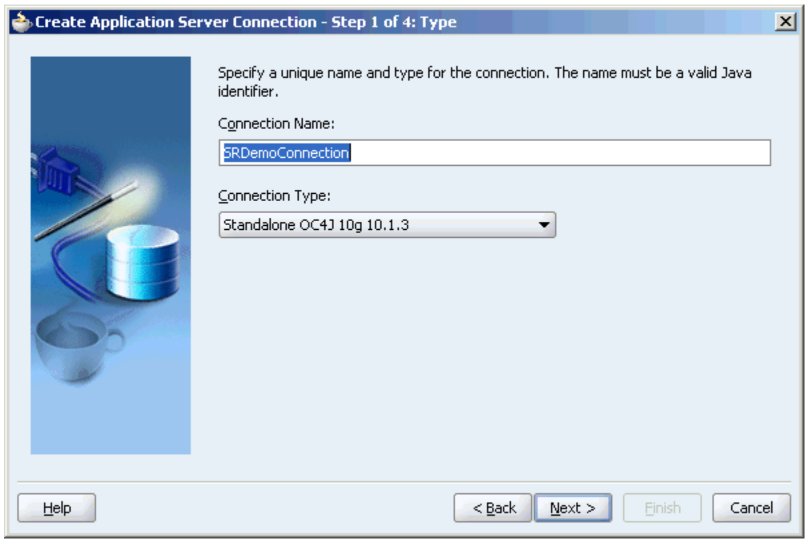
To create an application server connection to the preconfigured OC4J:

1. In Oracle JDeveloper, click the **Start WebCenter Preconfigured OC4J** icon. A `readme.html` file displays that contains the connection information for the preconfigured OC4J.
2. In the Connections Navigator, right-click **Connections** and select **New Application Server Connection**.

   Tip: If you cannot see the Connections Navigator, select **Connection Navigator** from the View menu.

3. If you are on the Welcome page of the Create Application Server Connect Wizard, click **Next** to display the Type page.
4. In the **Connection Name** field, shown in Figure 2–3, enter `SRDemoConnection`.
5. From the Connection Type list, select **Standalone OC4J 10g 10.1.3**.

Figure 2–3 Create Application Server Connection: Type



Specify a unique name and type for the connection. The name must be a valid Java identifier.

Connection Name: SRDemoConnection

Connection Type: Standalone OC4J 10g 10.1.3

6. Click **Next** to display the Authentication page.
7. Enter the Application Server Control Console administrator's user name and password for the relevant middle tier. Typically, the administrator's user name is `oc4jadmin` and the password is `welcome`.
8. Click **Next** to display the Connection page.
9. In the **Host Name** field, enter `localhost`.
10. In the **RMI Port** field, enter the port number to which the Remote Method Invokation (RMI) server binds to the OC4J server. The default value for the preconfigured OC4J is `22667`.
11. Click **Next** to display the Test page.
12. Click **Test Connection** to verify the connection. A success message should appear if everything is correct. If the test fails, you may need to review your connection information.
13. Click **Finish.**

Step 1: Building the Product Details Portlet (a JSR 168 Portlet)

To build a simple JSR 168 portlet, we've included steps for building a sample Service Requests Status portlet. Once you deploy this portlet, you will be able to add it to your My Acme page. To view a live demo version of a more highly interactive JSR 168 portlet using Ajax, simply run the completed version of the Service Request Demo.

For more information about JSR 168 portlets, refer to *Oracle WebCenter Framework Developer's Guide*.

Building a JSR 168 portlet involves the following tasks:

- Creating a JSR 168 Portlet
- Adding Portlet Logic to a JSR 168 Portlet
- Deploying a JSR 168 Portlet to an Application Server

Creating a JSR 168 Portlet

Oracle WebCenter Framework provides a Java Portlet Wizard that enables you to quickly create a simple implementation for each mode of your portlet. Let us use this wizard to get started creating our portlet.

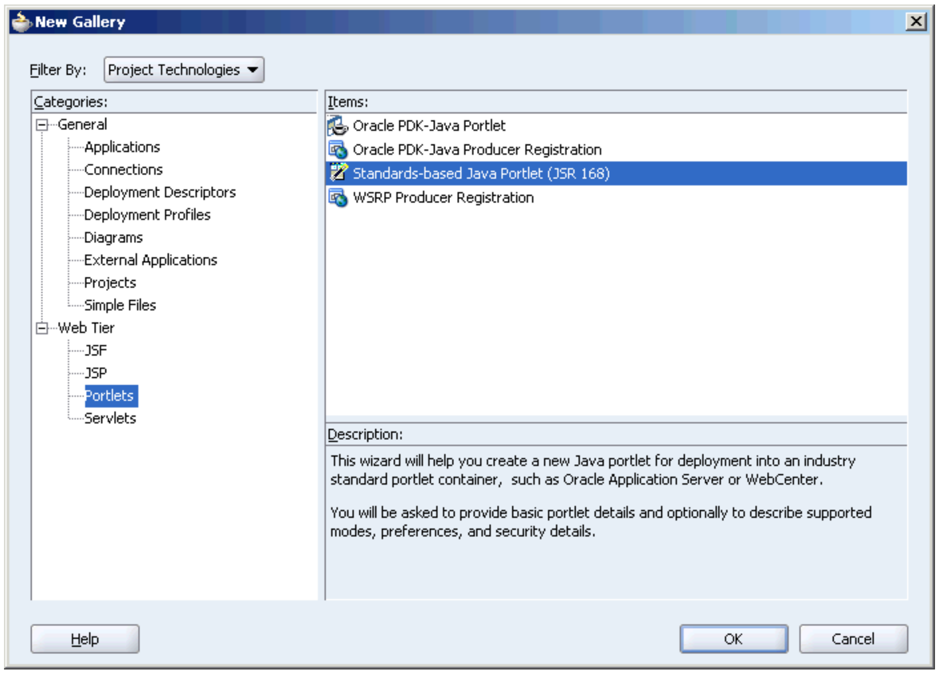
To create a JSR 168 portlet:

1. In Oracle JDeveloper, the Applications Navigator, within the **SRDemoSample_Starter** application, right-click the **ProdDetailsPortlet** project and select **New**.

   Tip: If this project does not exist, follow the instructions provided in Creating a Project for Your Portlets earlier in this chapter.

2. In the New Gallery, expand the **Web Tier** category and select **Portlets**.
3. In the Items list, shown in Figure 2–4, select **Standards-based Java Portlet (JSR 168)** to build a JSR 168 portlet.

Figure 2–4 Create a New JSR 168 Portlet



This wizard will help you create a new Java portlet for deployment into an industry standard portlet container, such as Oracle Application Server or WebCenter.

You will be asked to provide basic portlet details and optionally to describe supported modes, preferences, and security details.

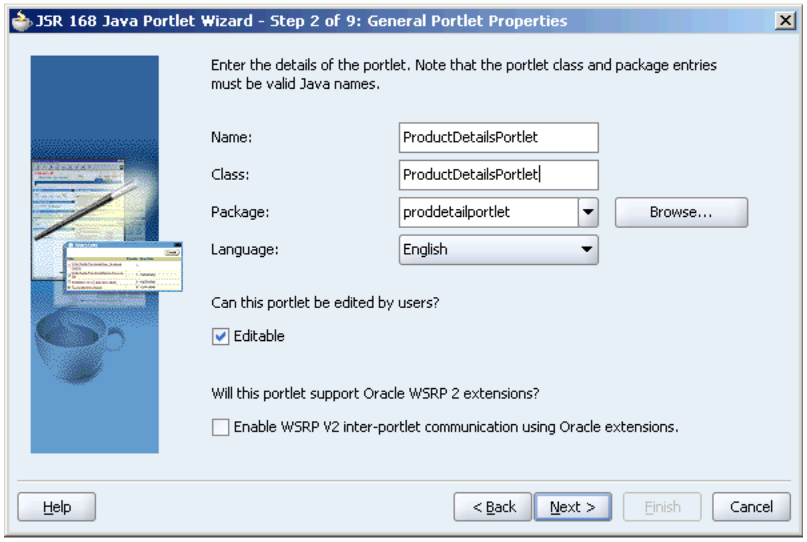
4. Click **OK** to display the JSR 168 Java Portlet Wizard.
5. If you are on the Welcome page of the wizard, click **Next** to display the Web Application page.
6. Select **Servlet 2.4\JSP 2.0 (J2EE 1.4)**.

   Tip: If a Web application has already been created for the project, you may not see this page of the wizard and instead go straight to the General Portlet Properties page.

7. Click **Next** to display the General Portlet Properties page.
8. In the **Name** and **Class** fields, shown in Figure 2–5, enter `ProductDetailsPortlet` and clear the **Enable WSRP V2-inter-portlet communication using Oracle extensions** check box.

The wizard creates a class with this name to represent the portlet.

Figure 2–5 JSR 168 Portlet General Portlet Properties



JSR 168 Java Portlet Wizard - Step 2 of 9: General Portlet Properties

Enter the details of the portlet. Note that the portlet class and package entries must be valid Java names.

Name: ProductDetailsPortlet

Class: ProductDetailsPortlet

Package: proddetailportlet

Language: English

Can this portlet be edited by users?

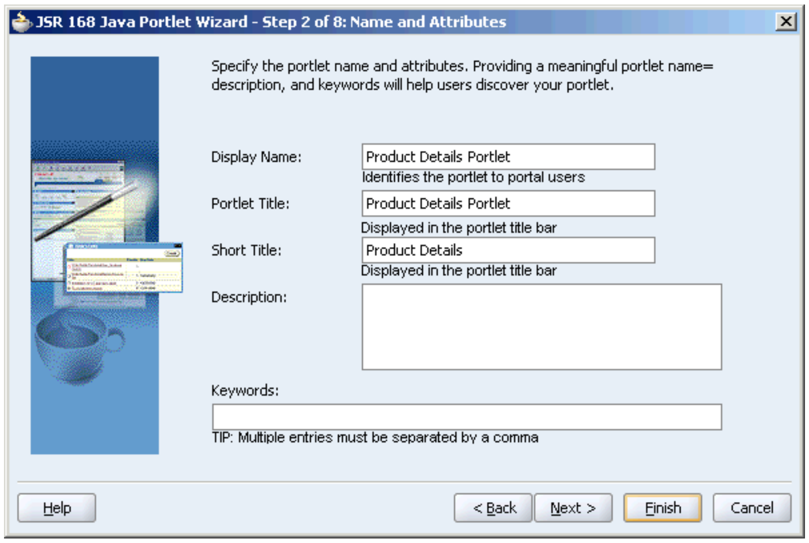
[✓] Editable

Will this portlet support Oracle WSRP 2 extensions?

[ ] Enable WSRP V2 inter-portlet communication using Oracle extensions.

9. Click **Next**.
10. In the **Display Name** field, shown in Figure 2–6, enter `Product Details Portlet`.
11. In the **Portlet Title** field, enter `Product Details Portlet`.
12. In the **Short Title** field, enter `Product Details`.

Figure 2–6 JSR 168 Portlet Name and Attributes



JSR 168 Java Portlet Wizard - Step 2 of 8: Name and Attributes

Specify the portlet name and attributes. Providing a meaningful portlet name= description, and keywords will help users discover your portlet.

Display Name: Product Details Portlet
Identifies the portlet to portal users

Portlet Title: Product Details Portlet
Displayed in the portlet title bar

Short Title: Product Details
Displayed in the portlet title bar

Description:

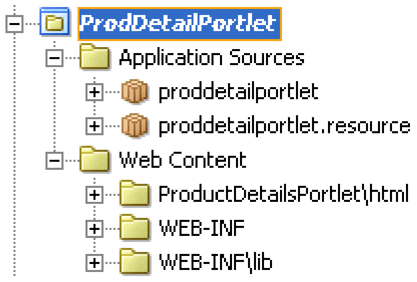
Keywords:

TIP: Multiple entries must be separated by a comma

13. Click **Finish** to accept the default settings for the rest of the portlet attributes.

    Your Product Details portlet now displays in the Applications Navigator as shown in Figure 2–7.

Figure 2–7 Product Details Portlet in the Applications Navigator



14. Save your work.

Creating a Web Service Proxy for Your JSR 168 Portlet

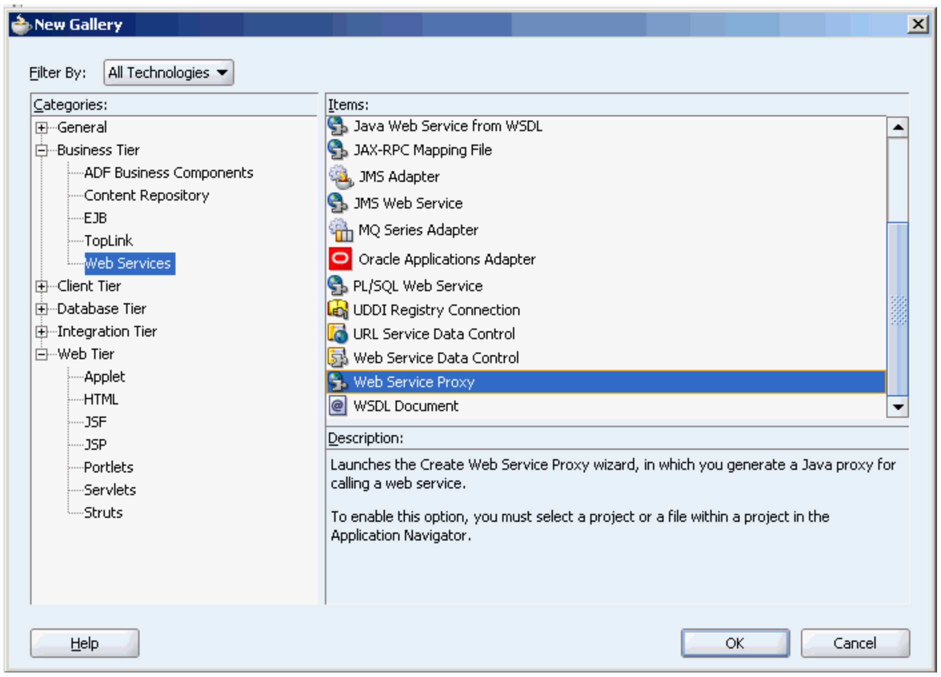
This JSR 168 portlet uses a Web Service as its data source. Before you can use a Web Service with a portlet in your application, you must first create a Web Service proxy for your portlet project.

To create a Web Service proxy:

1. Right-click the ProdDetailPortlet project that contains your ProductDetailsPortlet, then choose **New**.
2. Under the Business Tier node, shown in Figure 2–8, choose **Web Services**, then choose **Web Service Proxy.**

> **Tip:** If you do not see the Business Tier node in the New Gallery, you may need to Filter By All Technologies by selecting the option from the drop-down list.

Figure 2–8 New Gallery: Web Service Proxy



Launches the Create Web Service Proxy wizard, in which you generate a Java proxy for calling a web service.

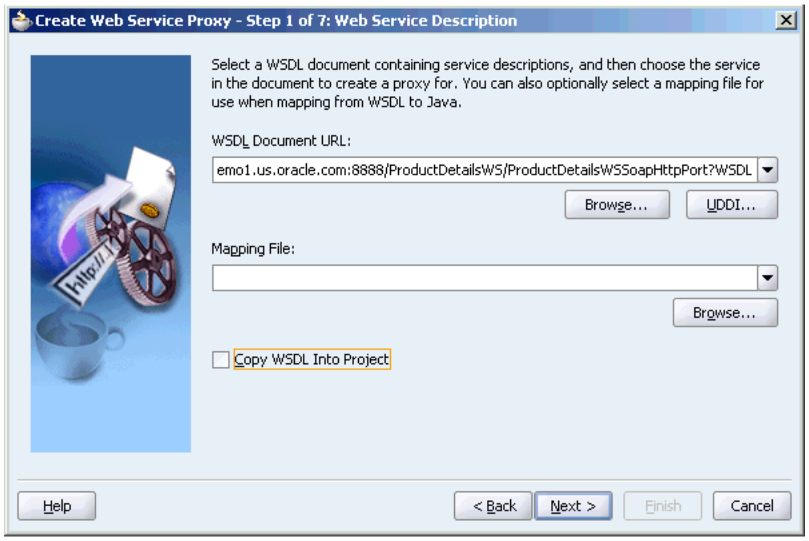
To enable this option, you must select a project or a file within a project in the Application Navigator.

3. Click **OK**.
4. On the Welcome page of the wizard, click **Next**.
5. On the Web Service Description page, shown in Figure 2–9, enter the WSDL of the Web Service , such as:

```
http://localhost:8888/ProductDetailsWS/ProductDetailsWSSoapHttpPo
rt?WSDL
```

Note: In this WSDL, localhost and port refer to the system where you deployed the Web Service. The Web Service is deployed in the SRDemo OC4J.

Figure 2–9 Set the Web Service Description


Create Web Service Proxy - Step 1 of 7: Web Service Description
Browse...
UDDI...
Browse...
Help
< Back
Next >
Finish
Cancel

Select a WSDL document containing service descriptions, and then choose the service in the document to create a proxy for. You can also optionally select a mapping file for use when mapping from WSDL to Java.

WSDL Document URL: emo1.us.oracle.com:8888/ProductDetailsWS/ProductDetailsWSSoapHttpPort?WSDL

Mapping File:

Copy WSDL Into Project

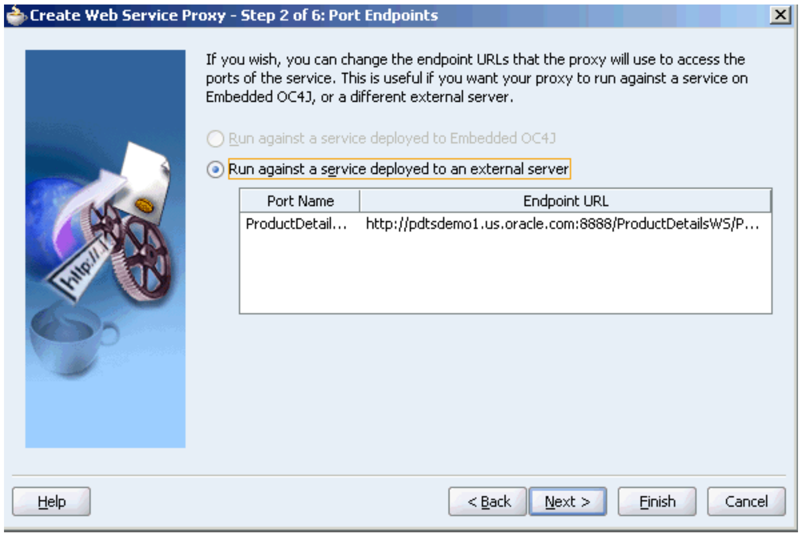
6. Click **Next**.

Note: You may need to click **Next** again if you do not see the Building Model dialog box display.

---

7. After Oracle JDeveloper builds the model for the Web Service, you should see the page shown in Figure 2–10, where you can validate the end point URL for your Web Service:

Figure 2–10 Validate the Web Service Port Endpoints



If you wish, you can change the endpoint URLs that the proxy will use to access the ports of the service. This is useful if you want your proxy to run against a service on Embedded OC4J, or a different external server.

Run against a service deployed to Embedded OC4J
[✓ Run against a service deployed to an external server]

Port Name	Endpoint URL
ProductDetail...	http://pdtsdemo1.us.oracle.com:8888/ProductDetailsWS/P...

8. Click **Finish** to accept the default values for the rest of the Web Service Proxy parameters.
9. Close the **ProductDetailsWSSSoapHttpPortClient.java** file.

Adding Portlet Logic to a JSR 168 Portlet

At the moment your portlet does not do very much. In fact, it just displays a Welcome message and allows the end user to personalize the portlet title. To make the portlet perform the specific function that you require, you need to extend the sample code with the business logic that implements the desired functionality and features.

First, your portlet will display a list of product types. The list is retrieved from a Web service. Later on, you will extend the portlet, allowing the end user to pick a product type, and select a particular product. When selecting a product, detailed information will display, including a brief description, and an image.

To add portlet logic to a JSR 168 portlet:

1. In the Applications Navigator, expand the **Web Content** node, right-click **view.jsp**, then select **Open** to open the file in the Visual Editor.

   Tip: To locate view.jsp, expand the following nodes under the SRDemoSample_Starter application: **Portlets > Web Content > ProductDetailsPortlet\html**.

2. At the bottom of the Visual Editor, click the **Source** tab to view the source code of the page.
3. Replace the existing code with the code shown in Example 2–1.

Example 2–1 Code for view.jsp

```
<%@ page contentType="text/html"
import="javax.portlet.*,java.util.*,portlets.ProductDetailsPortlet,portlets.resour
ce.ProductDetailsPortletBundle"%>

<%@ page contentType="text/html"
import="portlets.proxy.ProductDetailsWSSoapHttpPortClient,portlets.proxy.types.pro
ductdetailsws.types.ProductDetailsBean"%>

<%@ taglib uri="http://java.sun.com/portlet" prefix="portlet"%>

<portlet:defineObjects/>

<%
    String prodType = renderRequest.getParameter("prodType");
%>

<table cellpadding="10%">
 <tr>
  <td>
   <font class="portlet-section-header">
      Product Types
   </font>
  </td>
  <td>
   <font class="portlet-section-header">
      Products
   </font>
  </td>
  <td>
   <font class="portlet-section-header">
      Details
   </font>
  </td>
 </tr>
 <tr>
   <td>
<%
   ProductDetailsWSSoapHttpPortClient proxy = new
ProductDetailsWSSoapHttpPortClient();
   Vector productTypes =  proxy.getProductTypes();
       for (int i=0; i<productTypes.size(); i++)
       {
          PortletURL productURL = renderResponse.createRenderURL();
          productURL.setParameter ("prodType",
productTypes.elementAt(i).toString());
%>

         <font class="portlet-font">

           <a href="<%=productURL%>">
             <%= productTypes.elementAt(i) %>
           </a></font><br>
<%
       }
%>

  </td>
 </tr>
</table>
```

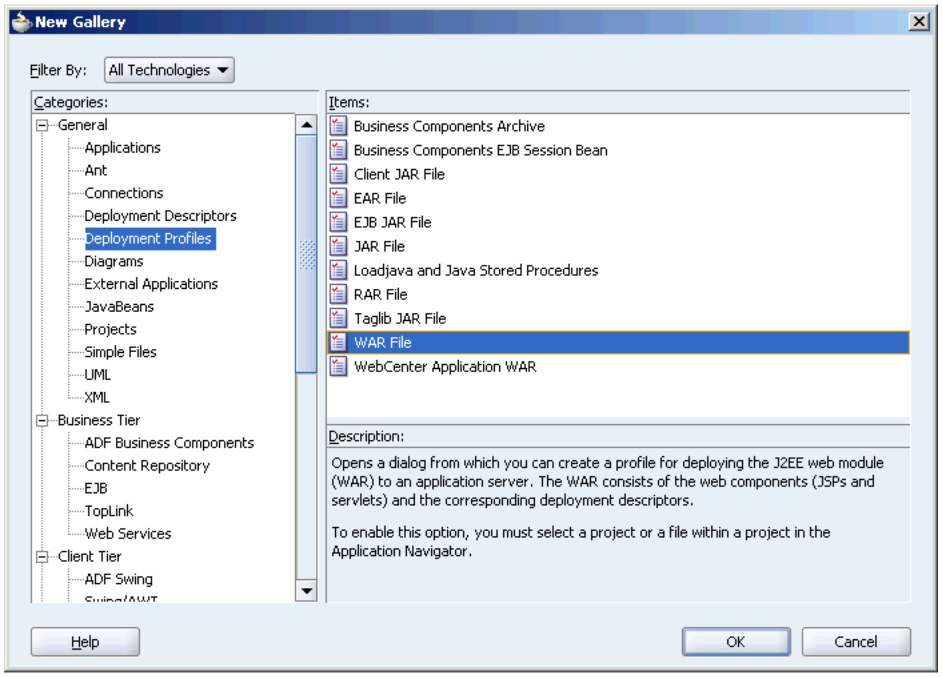
4. From the File menu, select **Save All**.

Deploying a JSR 168 Portlet to an Application Server

After you have finished building your portlet, you need package it and deploy to the portlet container.

1. In the Applications Navigator, right-click the **ProdDetailPortlet** project and select **New**.
2. In the New Gallery, expand the **General** category and select **Deployment Profiles**.
3. In the Items list, shown in Figure 2–11, select **WAR File**.

Figure 2–11 Create a WAR Deployment File



Description:

Opens a dialog from which you can create a profile for deploying the J2EE web module (WAR) to an application server. The WAR consists of the web components (JSPs and servlets) and the corresponding deployment descriptors.

To enable this option, you must select a project or a file within a project in the Application Navigator.

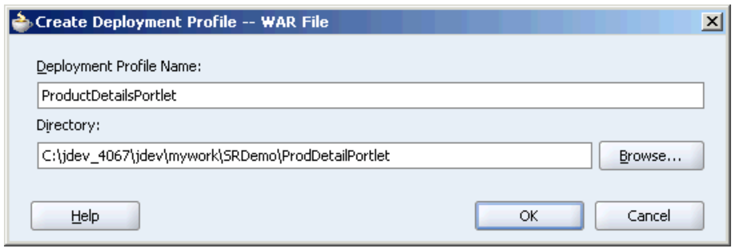
4. Click **OK** to display the Create Deployment Profile -- WAR File dialog box.

   Tip: You can also get to this dialog box by right-clicking the web.xml file in the Applications Navigator and selecting Create WAR Deployment File.

5. In the **Deployment Profile Name** field, shown in Figure 2–12, enter `ProductDetailsPortlet`.

6. Use the default directory.

Figure 2–12 Create Deployment Profile -- WAR File Dialog Box



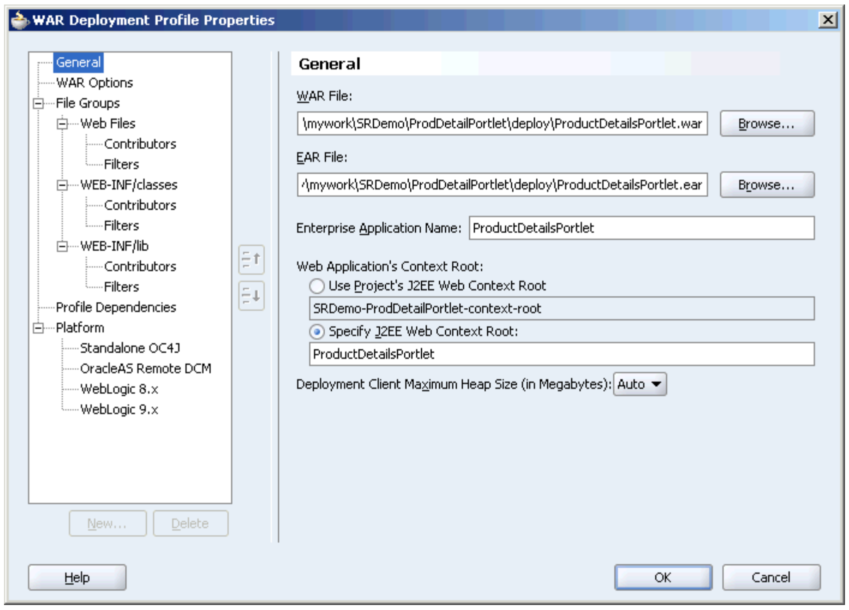
Deployment Profile Name: ProductDetailsPortlet

Directory: C:\jdev_4067\jdev\mywork\SRDemo\ProdDetailPortlet

7. Click **OK** to display the WAR Deployment Profile Properties dialog box.

8. Select **Specify J2EE Web Context Root**, and in the corresponding field, enter `ProductDetailsPortlet` as shown in Figure 2–13.

Figure 2–13 WAR Deployment Profile Properties



General

WAR File: \mywork\SRDemo\ProdDetailPortlet\deploy\ProductDetailsPortlet.war

EAR File: ^\mywork\SRDemo\ProdDetailPortlet\deploy\ProductDetailsPortlet.ear

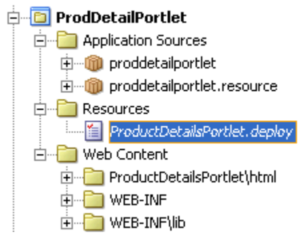
Enterprise Application Name: ProductDetailsPortlet

Web Application's Context Root:
( ) Use Project's J2EE Web Context Root
SRDemo-ProdDetailPortlet-context-root
(●) Specify J2EE Web Context Root:
ProductDetailsPortlet

Deployment Client Maximum Heap Size (in Megabytes): Auto

9. Click **OK.**
10. In the Applications Navigator, shown in Figure 2–14, expand the **Portlets** > **Resources** node to see the deployment file.

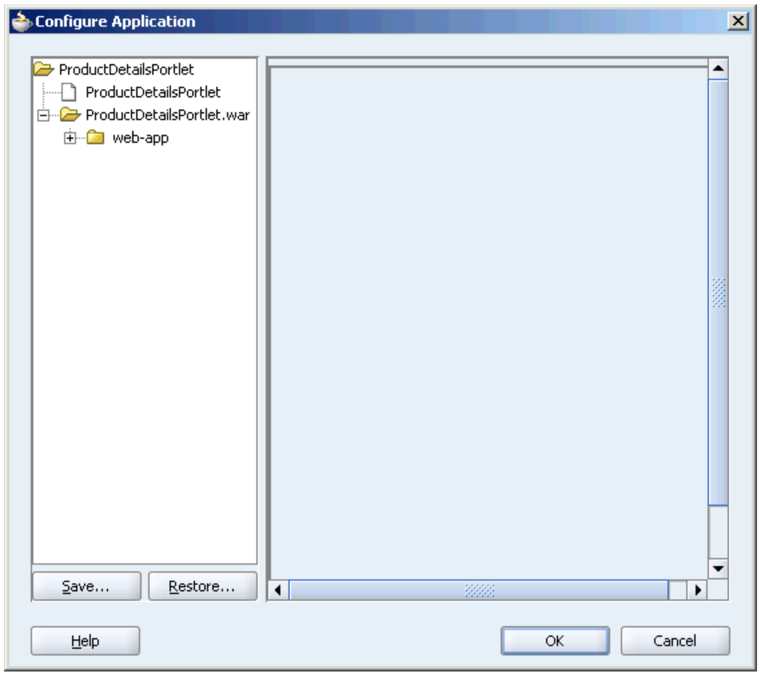
Figure 2–14 ProductDetailsPortlet.deploy in the Applications Navigator



11. From the File menu, choose **Save All**.
12. Right-click **ProductDetailsPortlet.deploy** and, from the Deploy to menu, shown in Figure 2–15, choose **SRDemoConnection**.

    The Configure Application dialog box displays.

Figure 2–15 Configure Application Dialog Box



Note: If SRDemoConnection does not appear in the menu, follow the steps provided in the step Creating an Application Server Connection to the Oracle WebCenter Preconfigured OC4J earlier in this chapter.

---

13. Click **OK.**
14. When the Deployment finished message displays in the Deployment Log, verify that no errors occurred.

Register the Producer

Now that you have created and deployed your portlet, you are ready to consume it in your application. You register the portlet producer with the application which enables you (or the application developer) to add the portlet to a page.

To register the producer:

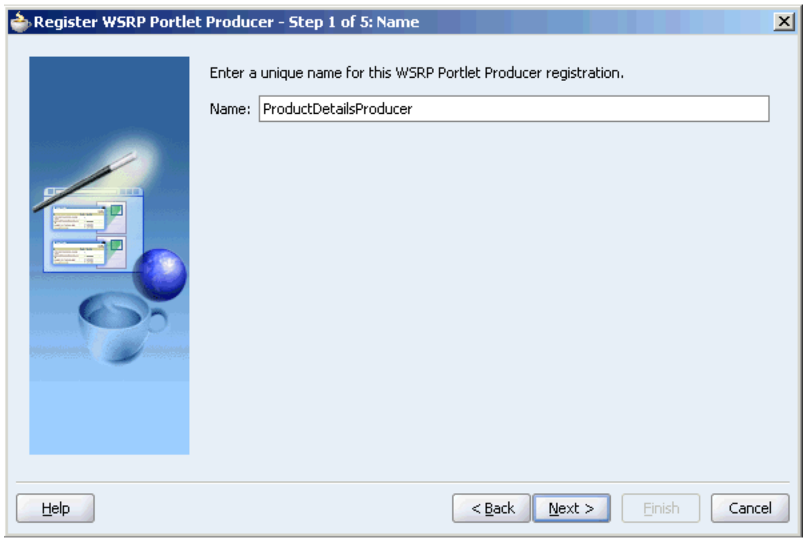
1. Since portlet producers are associated with your application, you can right-click the application or any project within the application to invoke. The easiest way, however, is to open the New Gallery from the **UserInterface** project. The technology scope of the UserInterface project is set up so that the producer registration wizards show up in your New Gallery.
2. Choose **New** from the pop-up menu.
3. Under **Web Tier > Portlets**, choose **WSRP Producer Registration**, then click **OK**.

   Tip: If you do not see the Business Tier node in the New Gallery, you may need to Filter By All Technologies by selecting the option from the drop-down list.

4. On the Welcome page, click **Next**.

5. On the Name page, shown in Figure 2–16, enter `ProductDetailsProducer` in the **Name** field, then click **Next**.

Figure 2–16 WSRP Producer Name



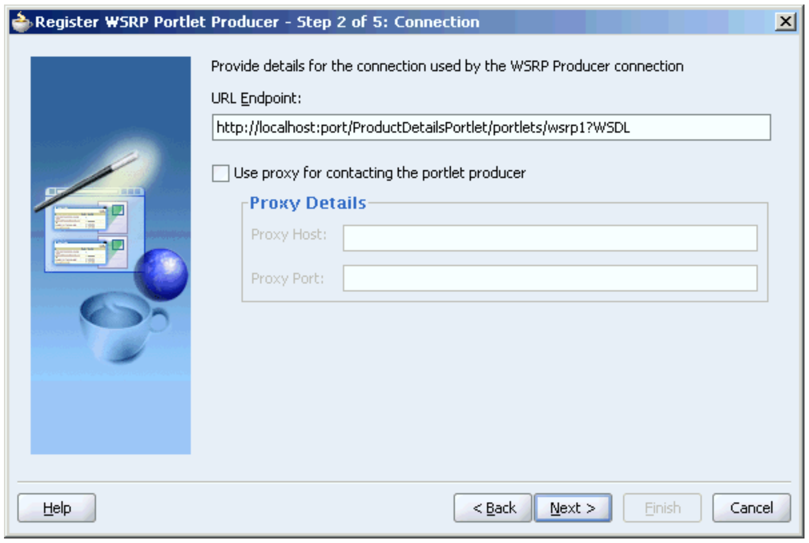
Enter a unique name for this WSRP Portlet Producer registration.

Name: ProductDetailsProducer

6. On the Connection page, shown in Figure 2–17, enter the URL for the WSRP container of your preconfigured OC4J, for example:

```
http://localhost:6688/ProductDetailsPortlet/portlets/wsrp1?WSDL
```

Figure 2–17 WSRP Producer URL



Provide details for the connection used by the WSRP Producer connection

URL Endpoint:

http://localhost:port/ProductDetailsPortlet/portlets/wsrp1?WSDL

☐ Use proxy for contacting the portlet producer

Proxy Details

Proxy Host:

Proxy Port:

7. Click **Next**.

8. After Oracle JDeveloper creates the connection, click **Finish**. When the success message displays, click **OK**.

Figure 2–18 Successful WSRP Producer Registration Message

Register WSRP Portlet Producer

Successfully registered WSRP Portlet Producer ProductDetailsProducer.

9. The portlet is now ready for you to add to a page.

Step 2: Building the Service Request Status Portlet (a PDK-Java Portlet)

In our Service Request application, we want to be able to display the current status of a particular service request, given the service request ID. We can build a portlet to display this information.

For more information about PDK-Java portlets, refer to the *Oracle WebCenter Framework Developer's Guide*.

Building a PDK-Java portlet involves the following tasks:

- Creating a PDK-Java Portlet and Producer
- Adding Portlet Logic to a PDK-Java Portlet
- Deploying a PDK-Java Portlet to an Application Server

Creating a PDK-Java Portlet and Producer

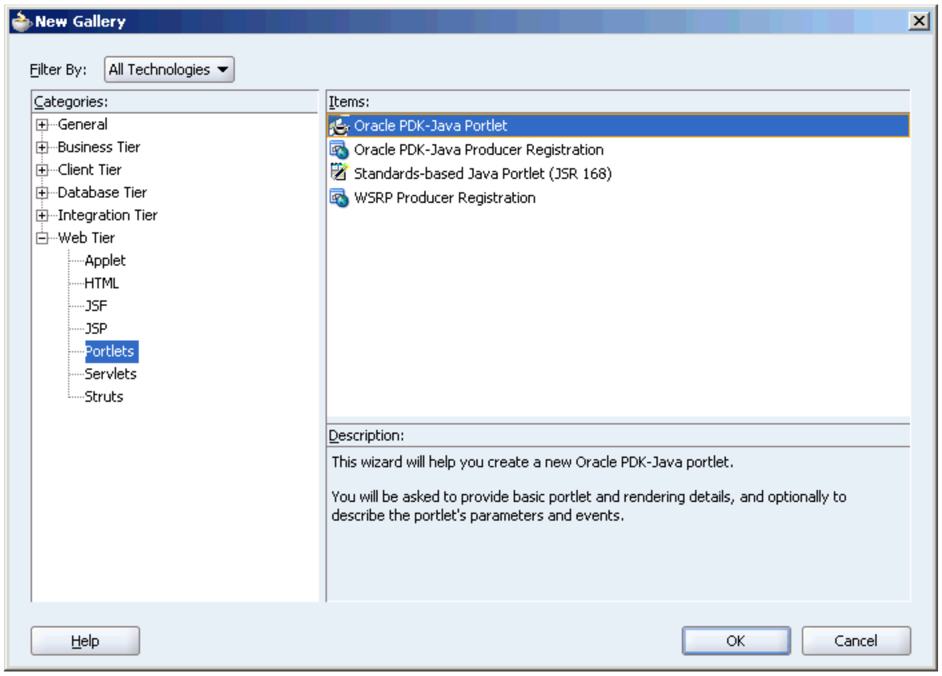
Oracle WebCenter Framework provides a Java Portlet Wizard that enables you to quickly create a simple implementation for each mode of your portlet. Let's use this wizard to get started creating our portlet.

1. In Oracle JDeveloper, in the Applications Navigator, expand the **SRDemoSample_Starter** application.
2. Right-click the **SRStatusPortlet** project and select **New**.

   Tip: If this project does not exist, follow the instructions provided in Creating a Project for Your Portlets earlier in this chapter.

3. In the New Gallery, expand the **Web Tier** category and select **Portlets**.
4. In the Items list, select **Oracle PDK-Java Portlet**, as shown in Figure 2–19

Figure 2–19 Create a New PDK-Java Portlet



This wizard will help you create a new Oracle PDK-Java portlet.

You will be asked to provide basic portlet and rendering details, and optionally to describe the portlet's parameters and events.

5. Click **OK** to display the Java Portlet Wizard.
6. If you are on the Welcome page of the wizard, click **Next** to display the Web Application page.
7. Click **Next** to display the Portlet Description page.
8. In the **Portlet Name** field, enter `SRStatusPortlet`.

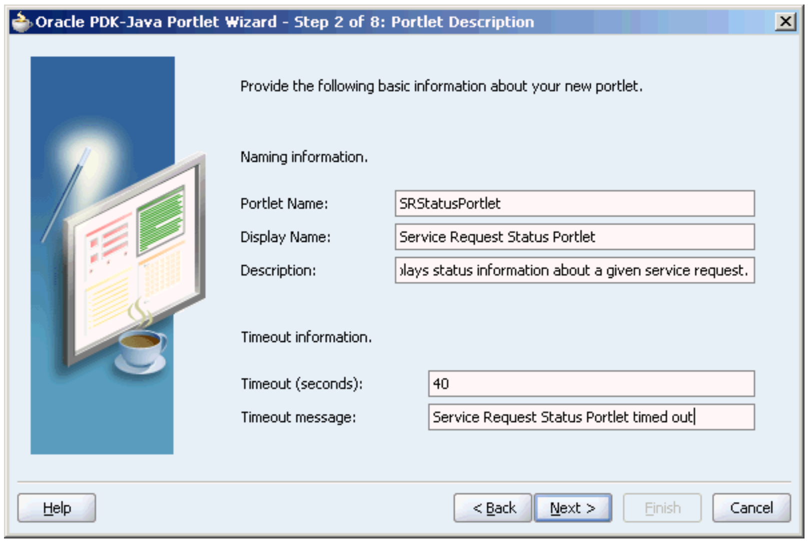
   This is an internal name only and is not exposed to end users.
9. In the **Display Name** field, enter `Service Request Status Portlet`.

   This is the name that appears in portlet selection lists, such as the Oracle JDeveloper Component Palette, where users choose which portlets to add to a page.
10. In the **Description** field, enter:

    ```
    This portlet displays status information about a given
    service request.
    ```
11. In the **Timeout (seconds)** field, leave the default value of 40.
12. In the **Timeout message** field, enter:

    ```
    Service Request Status Portlet timed out
    ```

Figure 2–20 Portlet Description Page


Oracle PDK-Java Portlet Wizard - Step 2 of 8: Portlet Description
Help
< Back
Next >
Finish
Cancel

Provide the following basic information about your new portlet.

Naming information.

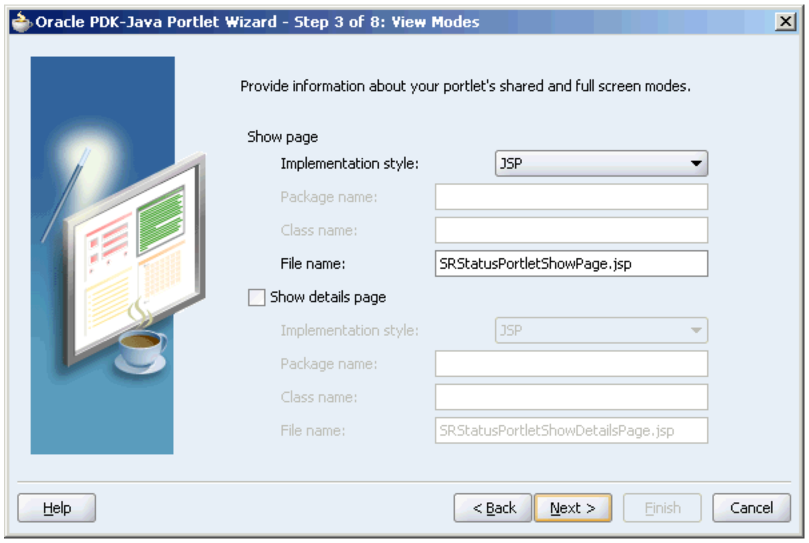
Portlet Name: SRStatusPortlet
Display Name: Service Request Status Portlet
Description: ›lays status information about a given service request.

Timeout information.

Timeout (seconds): 40
Timeout message: Service Request Status Portlet timed out

13. Click **Next** to display the View Modes page.
14. The **Show page** check box is selected by default. For Show mode:
    a. In the Implementation style list, make sure **JSP** is selected.
    b. In the File name field, keep the default value of `SRStatusPortletShowPage.jsp`.
15. We will not have a Show Details mode for this portlet, so leave the **Show details page** check box cleared.

Figure 2–21 Show Modes Page



Provide information about your portlet's shared and full screen modes.

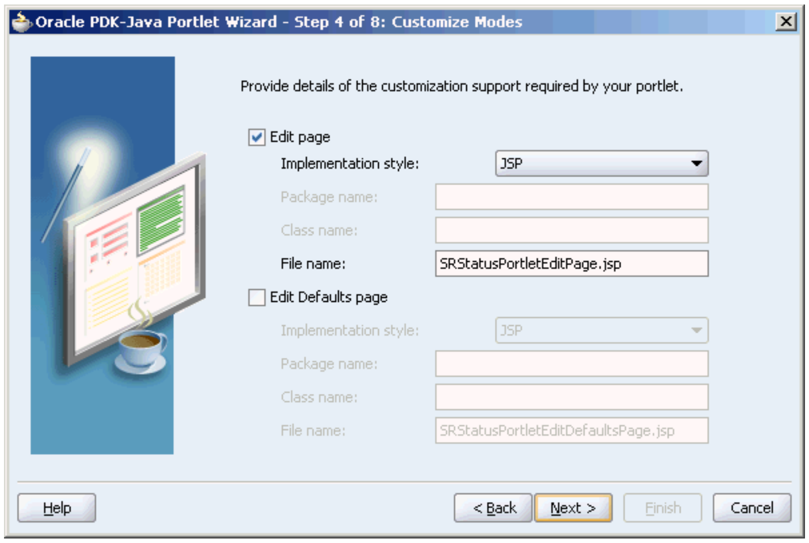
Show page
Implementation style: JSP
Package name:
Class name:
File name: SRStatusPortletShowPage.jsp

☐ Show details page
Implementation style: JSP
Package name:
Class name:
File name: SRStatusPortletShowDetailsPage.jsp

16. Click **Next** to display the Customize Modes page.
17. The Edit page check box is selected by default. For the Edit mode:
    a. In the Implementation style list, make sure that **JSP** is selected.
    b. In the **File name** field, keep the default value of `SRStatusPortletEditPage.jsp`.

18. We will not have an Edit Defaults mode for this portlet, so leave the Edit Defaults page check box cleared.

Figure 2–22 Customize Modes Page


Oracle PDK-Java Portlet Wizard - Step 4 of 8: Customize Modes

Provide details of the customization support required by your portlet.

- [✓] Edit page
  - Implementation style: JSP
  - Package name:
  - Class name:
  - File name: SRStatusPortletEditPage.jsp
- [ ] Edit Defaults page
  - Implementation style: JSP
  - Package name:
  - Class name:
  - File name: SRStatusPortletEditDefaultsPage.jsp

19. Click **Next** to display the Additional Modes page.

20. Select the **Help page** check box. For Help mode:

    a. From the **Implementation style** list, select **HTML File**.

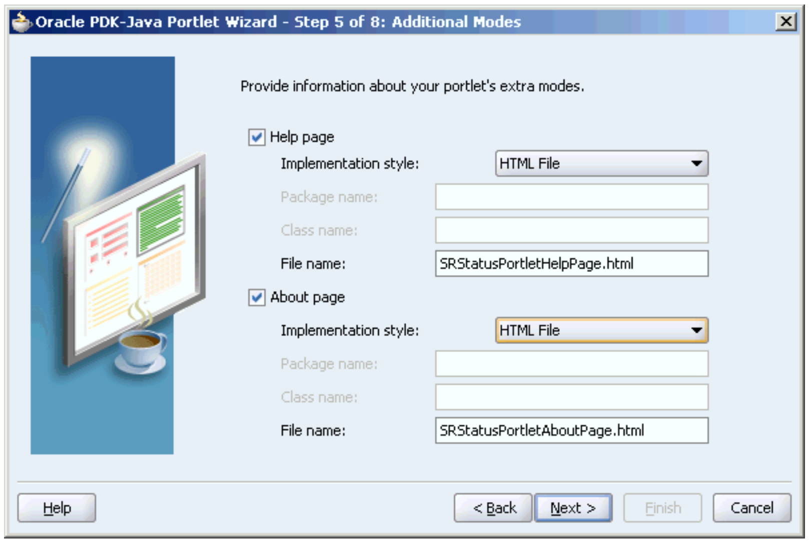
    b. In the **File name** field, keep the default value of `SRStatusPortletHelpPage.html`.

21. Select the **About page** check box. For About mode:

    a. From the Implementation style list, select **HTML File**.

    b. In the **File name** field, keep the default value of `SRStatusPortletAboutPage.html`.

Figure 2–23 Additional Modes Page

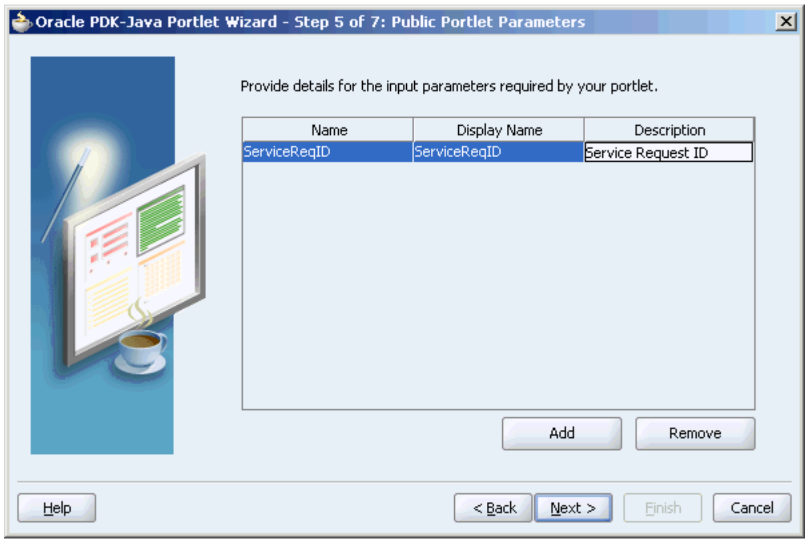

Oracle PDK-Java Portlet Wizard - Step 5 of 8: Additional Modes

Provide information about your portlet's extra modes.

- [✓] Help page
  - Implementation style: HTML File
  - Package name:
  - Class name:
  - File name: SRStatusPortletHelpPage.html
- [✓] About page
  - Implementation style: HTML File
  - Package name:
  - Class name:
  - File name: SRStatusPortletAboutPage.html

22. Click **Next** to display the Public Portlet Parameters page.
23. On the Public Portlet Parameters page, click **Add**.
24. Change the Name of the new parameter to `ServiceReqID`.
25. Change the Display Name of the new parameter to ServiceReqID.
26. Change the Description of the new parameter to `Service Request ID`.

Figure 2–24 Public Portlet Parameter

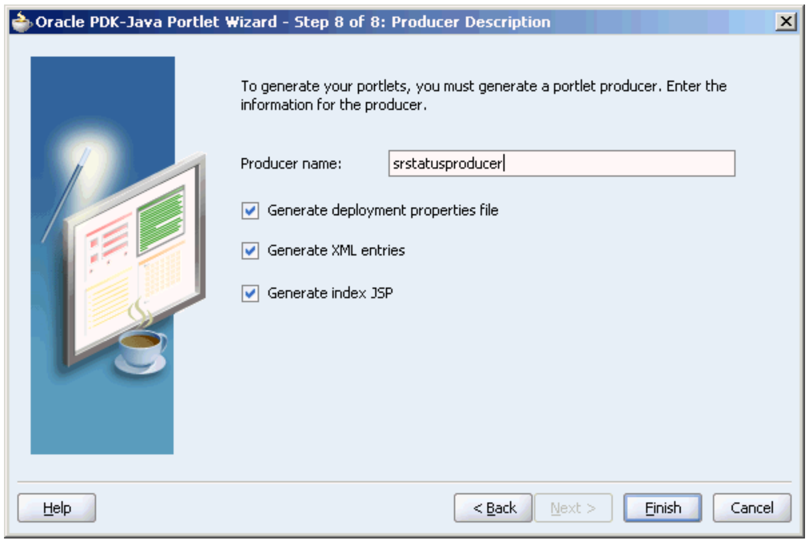

Oracle PDK-Java Portlet Wizard - Step 5 of 7: Public Portlet Parameters
Add
Remove
Help
< Back
Next >
Finish
Cancel

Provide details for the input parameters required by your portlet.

Name	Display Name	Description
ServiceReqID	ServiceReqID	Service Request ID

27. Click **Next** to display the Public Portlet Events page.
28. Click **Next** to display the Producer Description page.
29. In the **Producer name** field, enter `srstatusproducer`.
30. Ensure all the check boxes are selected: **Generate deployment properties file**, **Generate XML entries**, **Generate index JSP**.

Figure 2–25 Producer Description Page


Oracle PDK-Java Portlet Wizard - Step 8 of 8: Producer Description
Help
< Back
Next >
Finish
Cancel

To generate your portlets, you must generate a portlet producer. Enter the information for the producer.

Producer name: srstatusproducer

[✓] Generate deployment properties file

[✓] Generate XML entries

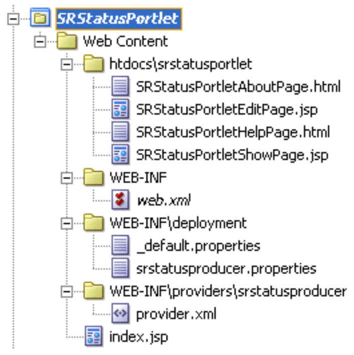
[✓] Generate index JSP

31. Click **Finish** to generate the files for your portlet. If you expand all the nodes under the SRStatusPortlet project in the Applications Navigator, you should see the following files:

    - A file for each portlet mode you selected:
        - `SRStatusPortletAboutPage.html`
        - `SRStatusPortletEditPage.jsp`
        - `SRStatusPortletHelpPage.html`
        - `SRStatusPortletShowPage.jsp`
    - `web.xml`
    - `_default.properties`
    - `srstatusproducer.properties`
    - `provider.xml`
    - `index.jsp`

    All of these files are required to deploy and run the portlet successfully, except for `index.jsp,` which is used by Oracle JDeveloper for testing purposes.

Figure 2–26 Files Generated for the Service Request Status Portlet



```
SRStatusPortlet
  Web Content
    htdocs\srstatusportlet
      SRStatusPortletAboutPage.html
      SRStatusPortletEditPage.jsp
      SRStatusPortletHelpPage.html
      SRStatusPortletShowPage.jsp
    WEB-INF
      web.xml
    WEB-INF\deployment
      _default.properties
      srstatusproducer.properties
    WEB-INF\providers\srstatusproducer
      provider.xml
    index.jsp
```

32. From the File menu, select **Save All**.

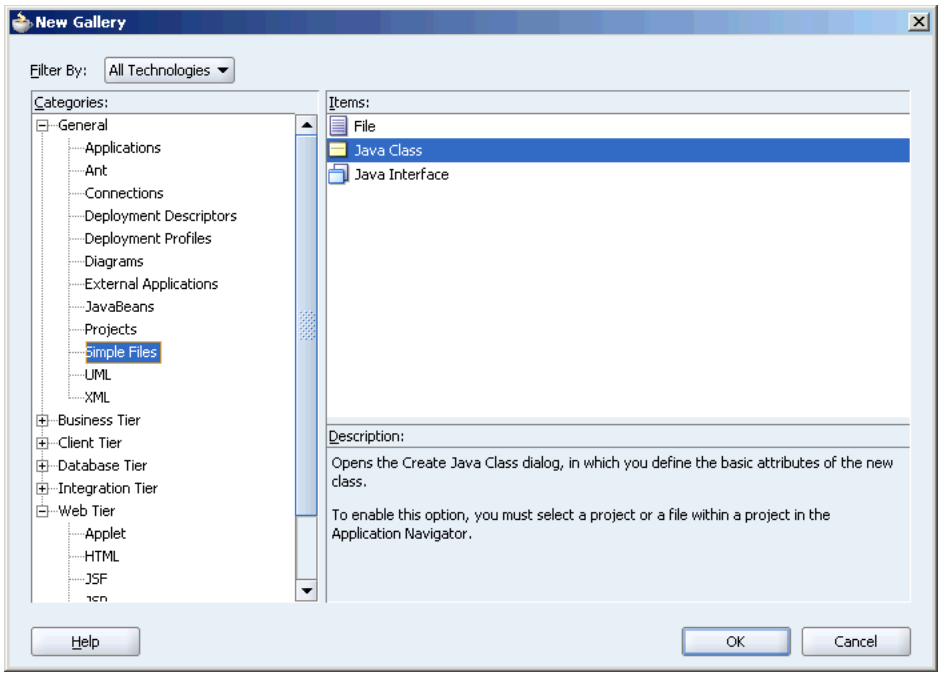
Adding Portlet Logic to a PDK-Java Portlet

At the moment, your portlet does not do very much. In fact, it just displays a welcome message and allows the end user to personalize the portlet title. To make the portlet perform the specific function that you require, you need to extend the sample code with wht business logic that implements the desired functionality and features.

The Service Request Portlet needs to query the SRDemo database to find out the status of a particular service request, given its service request ID, and then display that information in a table.

1. In the Applications Navigator, right-click the **SRStatusPortlet** project and select **New**.
2. In the New Gallery, expand the **General** category and select **Simple Files**.
3. In the Items list, select **Java Class**.

Figure 2–27 Create a Java Class

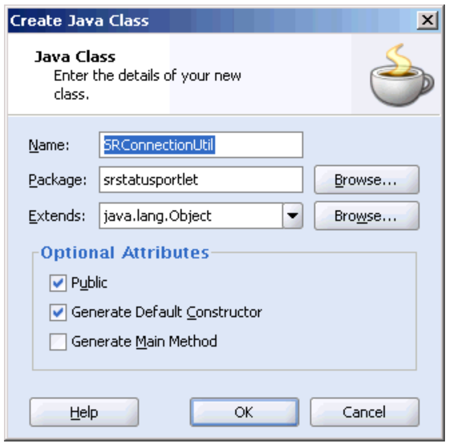


Opens the Create Java Class dialog, in which you define the basic attributes of the new class.

To enable this option, you must select a project or a file within a project in the Application Navigator.

4. Click **OK** to display the Create Java Class dialog box.
5. In the **Name** field, enter `SRConnectionUtil`.
6. Leave the rest of the options in this dialog box as their default values.

Figure 2–28 Create Java Class Dialog Box



Name: SRConnectionUtil
Package: srstatusportlet
Extends: java.lang.Object
☑ Public
☑ Generate Default Constructor
☐ Generate Main Method

7. Click **OK.**
8. Replace the generated code with the code shown in Example 2–2.

Example 2–2 Code to Create a Connection to the SRDemo Database

```
package srstatusportlet;

import java.sql.Connection;
import java.sql.SQLException;
import javax.naming.InitialContext;
import javax.naming.NamingException;

import javax.sql.DataSource;


public class SRConnectionUtil {
    public SRConnectionUtil() {
    }

    public static Connection getConnection ()
    {
      InitialContext ctx;
      DataSource ds;
      Connection conn = null;
      try
      {
        ctx = new InitialContext();
        ds = (DataSource) ctx.lookup("jdbc/SRDemoCoreDS");
        conn = ds.getConnection();
      }
      catch (NamingException ne)
      {
        conn = null;
      }
      catch (SQLException sqle)
      {
        conn = null;
      };
      return conn;
    }

}
```

This class creates a connection to the SRDemo database, against which we will execute a query in the portlet code.

9. Save the file.
10. If you have not already done so, create a database connection pointing to your srdemo schema as described in the `install.html` file located in the sample files you downloaded.
11. In the Applications Navigator, right-click **SRStatusPortletShowPage.jsp** and select **Open**.
12. Select the Source tab to view the source code for the page.
13. Select the existing code and replace it with the code shown in Example 2–3.

Example 2–3 Code for the Service Request Status Portlet

```
<%@page contentType="text/html; charset=windows-1252"
        import="oracle.portal.provider.v2.render.PortletRenderRequest"
        import="oracle.portal.provider.v2.http.HttpCommonConstants"
        import="oracle.portal.provider.v2.ParameterDefinition"
        import="srstatusportlet.SRConnectionUtil"
        import="java.sql.Connection"
        import="java.sql.Statement"
        import="java.sql.ResultSet"
        import="java.sql.SQLException"
%>
```

```
<%
   PortletRenderRequest pReq = (PortletRenderRequest)
      request.getAttribute(HttpCommonConstants.PORTLET_RENDER_REQUEST);

   // Reading the service request ID, as a parameter
   String SRID = pReq.getParameter("ServiceReqID");

   // Creating the database connection
   Connection conn = SRConnectionUtil.getConnection();

   // Testing if the parameter has been wired. If no parameter is passed
   // it is set to 104, by default.
   if (SRID == null)
   {
     %>
     Please map the service request ID to this portlet's input parameter.
     <%
     SRID = "104";
   }

   if (conn == null) {
    %>
    Couldn't connect to the database.
    <%
   }
   else
     try
     {
      Statement stmt = conn.createStatement();
      // Constructing the SELECT statement
      String query =
          "SELECT svr_id, status, " +
          "users1.first_name || ' ' || users1.last_name createdby, " +
          "users2.first_name || ' ' || users2.last_name assignedto, " +
          "TO_CHAR(assigned_date, 'Dy, Mon DD, YYYY') " +
          "FROM service_requests, users users1, users users2 " +
          "WHERE svr_id = " + SRID + " AND " +
          "users1.user_id = service_requests.created_by AND " +
          "users2.user_id = service_requests.assigned_to";

      // Executing the query.
      ResultSet rs = stmt.executeQuery(query);
      // Stepping through the result set.
      while (rs.next())
      {
%>
<table>
 <tr>
  <td>
   <font class="PortletText1">
     Service Request ID:
   </font>
  </td>
  <td>
   <font class="PortletHeading1">
     <%=rs.getInt(1)%>
   </font>
  </td>
 </tr>
 <tr>
  <td>
   <font class="PortletText1">
     Status:
   </font>
  </td>
  <td>
   <font class="PortletHeading1">
     <%=rs.getString(2)%>
   </font>
  </td>
 </tr>
 <tr>
  <td>
   <font class="PortletText1">
     Created on:
   </font>
  </td>
  <td>
   <font class="PortletHeading1">
     <%=rs.getString(5)%>
   </font>
  </td>
 </tr>
 <tr>
  <td>
   <font class="PortletText1">
     Created By:
   </font>
  </td>
  <td>
   <font class="PortletHeading1">
     <%=rs.getString(3)%>
   </font>
  </td>
 </tr>
 <tr>
  <td>
   <font class="PortletText1">
     Assigned To:
   </font>
  </td>
  <td>
   <font class="PortletHeading1">
     <%=rs.getString(4)%>
   </font>
  </td>
 </tr>
</table>

<%
        } //while
      } // try
      catch (SQLException sqle)
      {
        System.out.println ("DB Connection established successfully but ran into
an issue while working with the DB.");
        System.out.println (sqle);
      }
      conn.close();
```

```
%>
```

This code first calls the Java class we created earlier to connect to the database. Then it sets the service request ID based on a parameter passed to it (if no parameter is passed, the service request ID is set to a default value of 100). Using this service request ID, the portlet then queries the SRDemo database to find the status of the service request. Finally the results of this query are printed in an HTML table.

14. Save the file.
15. Right-click `SRStatusPortletAboutPage.html` and select **Open**.
16. In the Design view, enter the following text:

    ```
    Service Request Status Portlet
    MyAcme, Copyright
    ```

17. Save the file.
18. Right-click **SRStatusPortletHelpPage.html** and select **Open**.
19. In the Design view, enter the following text:

    ```
    This portlet displays status information about a given
    service request, based on a parameter (service request ID) it
    receives. If the portlet does not work as expected, it may
    not have been wired properly.
    ```

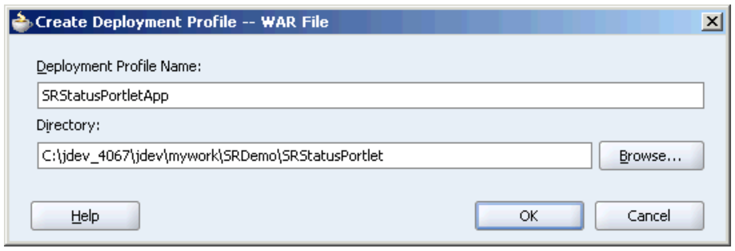
20. Save the file.

Deploying a PDK-Java Portlet to an Application Server

After you have finished building your portlet, you are ready to deploy it to your preconfigured OC4J.

1. In the Applications Navigator, locate the **web.xml** file under **SRStatusPortlet > Web Content > WEB-INF**.
2. Right-click the **web.xml** and select **Create WAR Deployment Profile**
3. In the **Deployment Profile Name** field, enter `SRStatusPortletApp`.
4. Use the default directory.

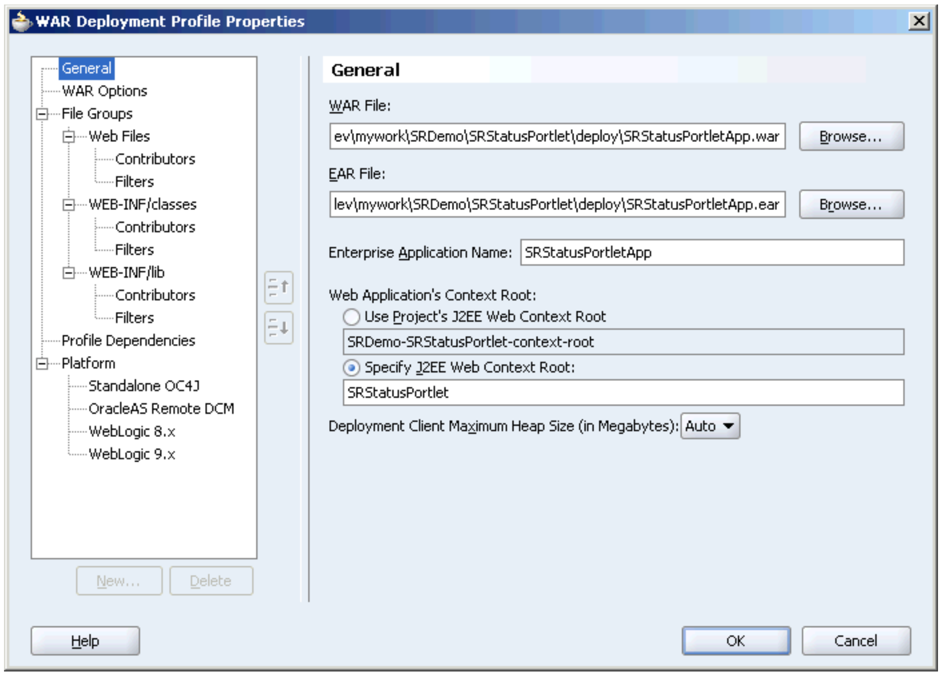
Figure 2–29 Create Deployment Profile -- WAR File Dialog Box



Deployment Profile Name: SRStatusPortletApp
Directory: C:\jdev_4067\jdev\mywork\SRDemo\SRStatusPortlet

5. Click **OK** to display the WAR Deployment Profile Properties dialog box.
6. Select **Specify J2EE Web Context Root**, and in the corresponding field, enter `SRStatusPortlet`.

Figure 2–30 WAR Deployment Profile Properties Dialog Box



General

WAR File: ev\mywork\SRDemo\SRStatusPortlet\deploy\SRStatusPortletApp.war

EAR File: lev\mywork\SRDemo\SRStatusPortlet\deploy\SRStatusPortletApp.ear

Enterprise Application Name: SRStatusPortletApp

Web Application's Context Root:

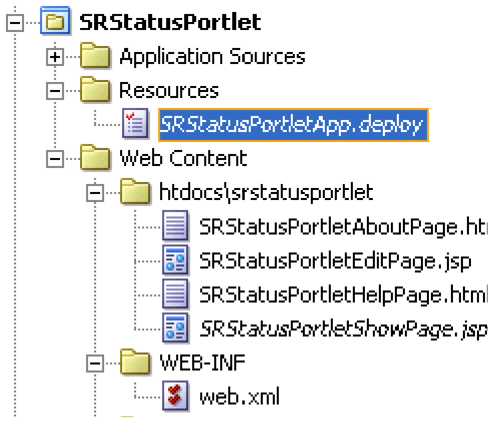
( ) Use Project's J2EE Web Context Root: SRDemo-SRStatusPortlet-context-root

(●) Specify J2EE Web Context Root: SRStatusPortlet

Deployment Client Maximum Heap Size (in Megabytes): Auto

7. Click **OK.**
8. In the Applications Navigator, expand the **Resources** node to see the deployment file.

Figure 2–31 SRStatus Portlet Deployment File



```
SRStatusPortlet
  Application Sources
  Resources
    SRStatusPortletApp.deploy
  Web Content
    htdocs\srstatusportlet
      SRStatusPortletAboutPage.ht
      SRStatusPortletEditPage.jsp
      SRStatusPortletHelpPage.html
      SRStatusPortletShowPage.jsp
    WEB-INF
      web.xml
```

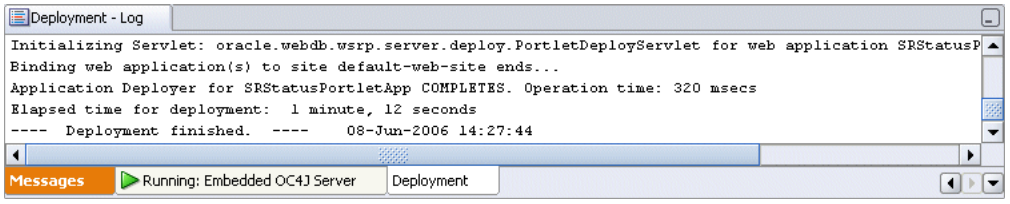
9. From the File menu, choose **Save All**.
10. Right-click **SRStatusPortletApp.deploy** and from the Deploy to menu, select **SRDemoConnection**.

    The Configure Application dialog box displays.

> **Note:** If SRDemoConnection does not appear in the menu, follow the steps provided in Creating an Application Server Connection to the Oracle WebCenter Preconfigured OC4J earlier in this chapter.

11. You can accept the defaults here, so click **OK.**
12. When the Deployment finished message displays in the Deployment Log, verify that no errors occurred.

Figure 2–32 Deployment Log



```
Initializing Servlet: oracle.webdb.wsrp.server.deploy.PortletDeployServlet for web application SRStatusP
Binding web application(s) to site default-web-site ends...
Application Deployer for SRStatusPortletApp COMPLETES. Operation time: 320 msecs
Elapsed time for deployment:  1 minute, 12 seconds
----  Deployment finished.  ----    08-Jun-2006 14:27:44
```

13. Register the producer following the same steps you used in "Register the Producer" for the JSR 168 portlet, using the URL:

    ```
    http://localhost:6688/SRStatusPortlet/providers
    ```

    The JSR 168 portlet and PDK-Java portlet you created are now ready for you to add to your application.

Summary

In this chapter, you learned how to build a JSR 168 portlet and a PDK-Java portlet and how to deploy them for use with a WebCenter application. You can now register the producer with an application and add the portlet to a page. You will do just that in Chapter 7, "Building a Page and Adding Components". Before you move onto building the page and adding components however, you can continue to Chapter 3, "Setting Up Skins".

3 Setting Up Skins

In this chapter, you will configure your application to use the skins you downloaded as part of the demo application, so that they are available for selection when running your application.

Before you can perform the tasks in this chapter, you must have extracted the skins from the `skins.zip` file to the appropriate folder by performing the steps in the `install.html` document available as part of the `SRDemo_App_Download.ZIP`. This ZIP file contains two skins, `myCompany` and `limerine`, and the corresponding image files you may require. By default, the `srdemo` skin is available as part of the `SRDemoSample_Starter` application.

This chapter includes the following sections:

- Step 1: Verifying that the Skins are Registered
- Step 2: Verifying that Your Application is Configured to Use the New Skins

Step 1: Verifying that the Skins are Registered

To apply the new skins, `myCompany` and `limerine` for your application, you must first ensure that these skins are registered with your WebCenter application. For this, you must ensure that the `adf-faces-skins.xml` file, which contains information about all the skins available for the application, contains entries for these three skins.

For more information about skins, including how to create new skins, refer to *Oracle Application Development Framework Developer's Guide*.

To verify that the three new skins are registered with your application, perform the following steps:

1. In the Applications Navigator, expand the **UserInterface** project.
2. Expand the **Web Content** folder, then the **WEB-INF** folder.
3. Right-click **adf-faces-skins.xml** and select **Open**.
4. Ensure that the code in bold text in Example 3–1 exists in the file.

Example 3–1 Entries for Skins in adf-faces-skins.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<skins xmlns="http://xmlns.oracle.com/adf/view/faces/skin">

    <skin>
        <id>srdemo.desktop</id>
        <family>srdemo</family>
        <render-kit-id>oracle.adf.desktop</render-kit-id>
        <style-sheet-name>skins/srdemo/srdemo.css</style-sheet-name>
    </skin>
    <skin>
        <id>mycompany.desktop</id>
        <family>mycompany</family>
        <render-kit-id>oracle.adf.desktop</render-kit-id>
        <style-sheet-name>skins/mycompany/myCompanySkin.css</style-sheet-name>
    </skin>
    <skin>
        <id>limerine.desktop</id>
        <family>limerine</family>
        <render-kit-id>oracle.adf.desktop</render-kit-id>
        <style-sheet-name>skins/limerine/limerine.css</style-sheet-name>
    </skin>
</skins>
```

5. Close the file.

Step 2: Verifying that Your Application is Configured to Use the New Skins

To use a different skin, the `<skin-family>` tag in the `adf-faces-config.xml` file must have been updated with the family name of the desired skin. The `adf-faces-config.xml` file contains information about the skin to be used at run time. In most cases, the skin family name is hardcoded in the configuration file.

However, in our application we want the administrator to be able to select a skin at run time. For this, the skin family name in the `adf-faces-config.xml` file must use expression language (EL), instead of being hardcoded. This section describes the procedure to verify this.

To verify that the `adf-faces-config.xml` file uses an EL for the skin family name, perform the following steps:

1. In the Applications Navigator, expand the *`<SRDemoSample>`* application.
2. Expand the **UserInterface** project.
3. Expand the **Web Content** node, then the **WEB-INF** node.
4. Right-click **adf-faces-config.xml** and select **Open**.
5. In the Structure pane, select **skin-family**.
6. In the Property Inspector, ensure that `skin-family` is set to `#{skinBean.currentSkin}`.

Summary

In this chapter, you learned how to register new skins with your application and configure the application to use the new skins. You can now provide users the option to change the skin at run time. Chapter 11, "Building a Site Administration Page" describes the steps involved in enabling this type of customization.

4 Setting Up Your Content Repository

In this chapter, you will learn how to set up your content repository from a content directory located on your local drive. Later, you will integrate content from this repository with the JSPX pages that you will build, as you implement procedures in other chapters.

To setup your content repository, you will configure your local directory as a File System Java Content Repository (JCR) data control that provides you with `search`, `advancedSearch`, `getURI`, and `getItems` methods. The `getURI` and `getItems` methods enable you to add content as links, tables, and hierarchical trees. The `search` and `advancedSearch` methods enable you to provide the search and advanced search functionality for the content that you add.

In this chapter, you will perform the following steps:

- Step 1: Setting Up a Content Directory for the Sample Content
- Step 2: Creating a Content Project
- Step 3: Configuring a JCR Data Control

Step 1: Setting Up a Content Directory for the Sample Content

Before configuring the File System data control, set up the content directory on your local drive by performing the steps in section titled "Set Up the Sample Content" in the `install.html` file located in the sample files you downloaded.

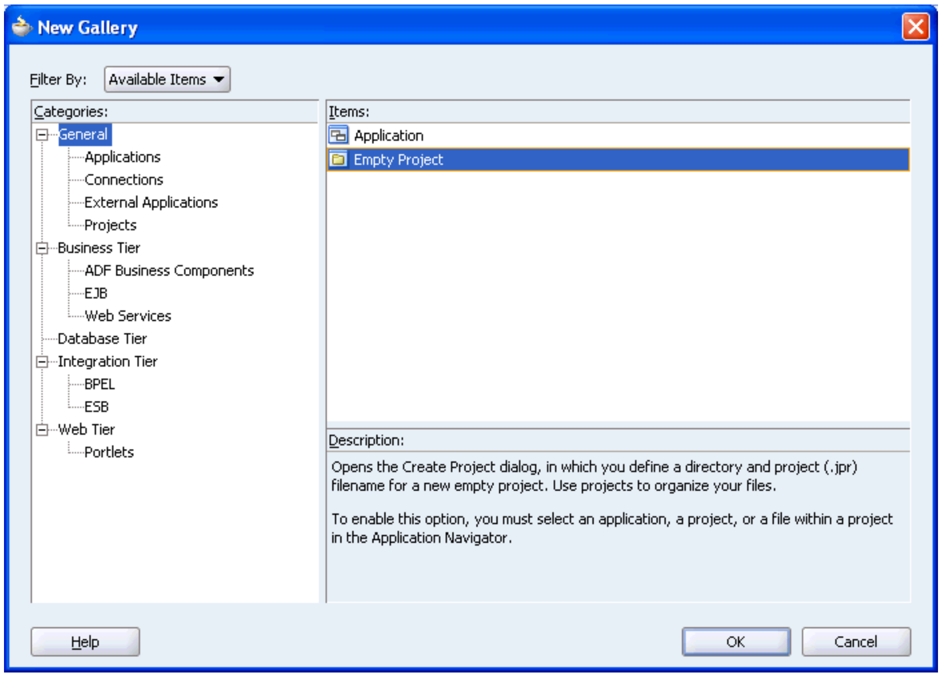
Step 2: Creating a Content Project

In this section, you will create a project for which you will configure the data control in the next step.

To create your project, perform the following steps:

1. In Oracle JDeveloper, go to the Applications Navigator. Under Applications, right-click **SRDemoSample_Starter** and select **New**. The New Gallery dialog box is displayed.
2. Under Items, select **Empty Project**, as shown in Figure 4–1 and click **OK**. The Create Project dialog box is displayed.

Figure 4–1 The New Gallery Dialog Box - Empty Project



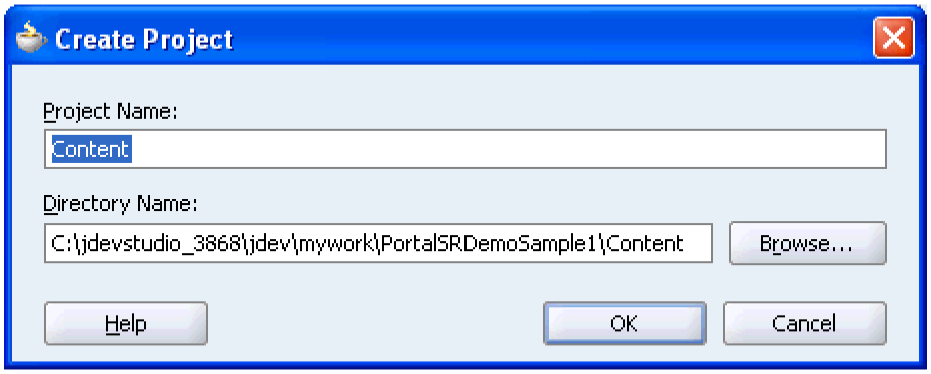
Description:

Opens the Create Project dialog, in which you define a directory and project (.jpr) filename for a new empty project. Use projects to organize your files.

To enable this option, you must select an application, a project, or a file within a project in the Application Navigator.

3. Enter **Content** in the Project Name field, as shown in Figure 4–2 and click **OK**. The Content project displays in the Applications Navigator, as shown in Figure 4–3.

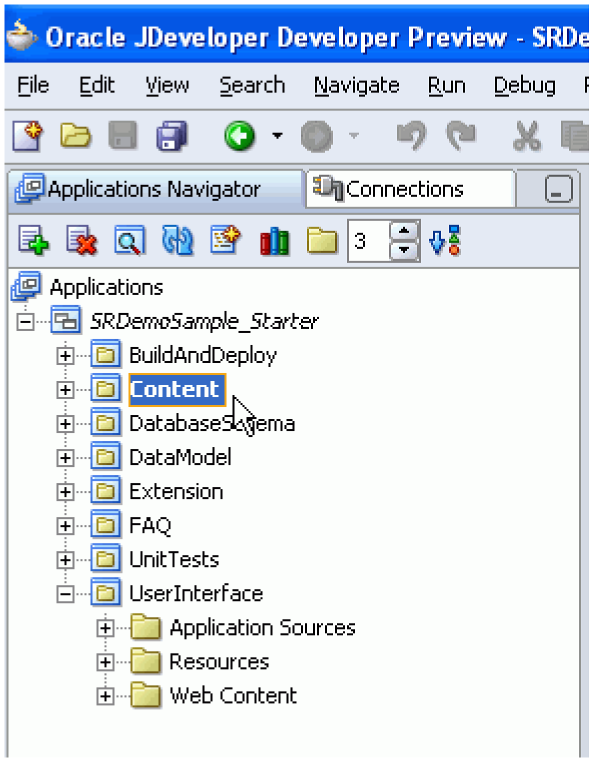
Figure 4–2 New Project



Project Name: Content

Directory Name: C:\jdevstudio_3868\jdev\mywork\PortalSRDemoSample1\Content

Figure 4–3 Applications Navigator - Content Project



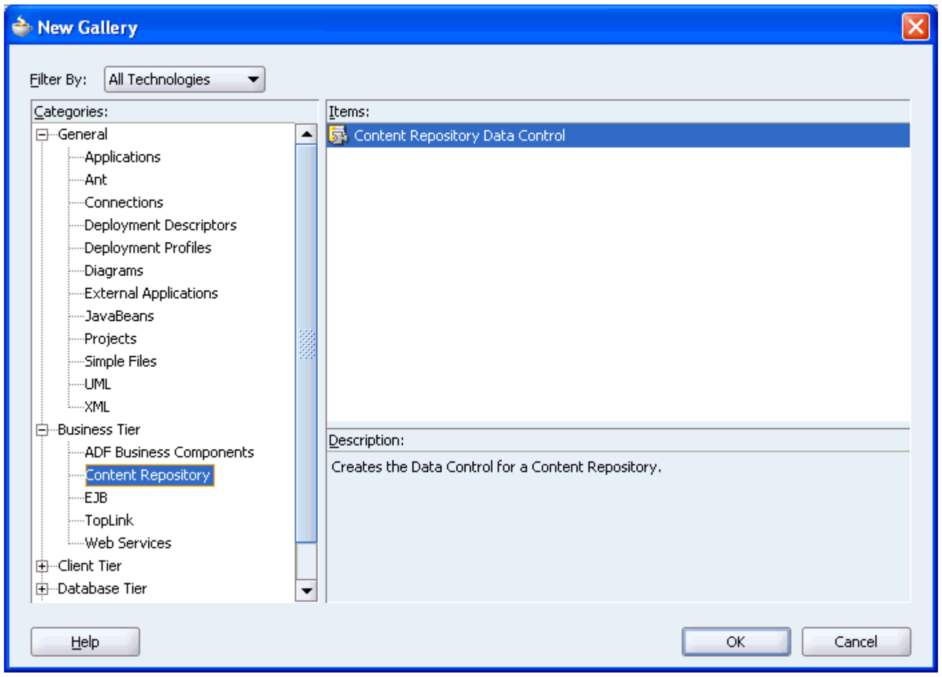
Step 3: Configuring a JCR Data Control

In this section, you will configure a File System JCR data control that can access and publish content located on your local drive.

To configure your File System data control, perform the following steps:

1. Under the Applications Navigator, right-click **Content** and select **New**. The New Gallery dialog box is displayed.
2. Under Categories, expand the **Business Tier** node and select **Content Repository**. Then, under Items select **Content Repository Data Control** as shown in Figure 4–4, and click **OK**. The Create Data Control dialog box is displayed.

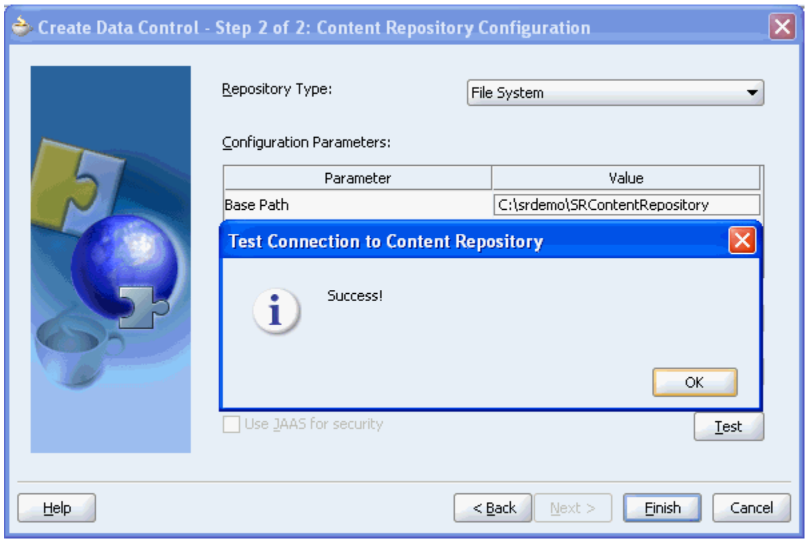
Figure 4–4 The New Gallery Dialog Box - Content Repository Data Control



Creates the Data Control for a Content Repository.

3. In the Create Data Control dialog box, click **Next** to skip the Welcome page.
4. On step 1, enter `SRContentRepository` as the name for the data control, and then click **Next**.
5. On step 2, select **File System** from the **Repository Type** box.
6. In the **Base Path** field, enter the path to the folder in which your content is placed, that is, `C:\srdemo\SRContentRepository`.
7. Click the **Test** button to check the connection. You should see the `Success!` message, as shown in Figure 4–5.

Figure 4–5 Test Connection to the SRContentRepository


Create Data Control - Step 2 of 2: Content Repository Configuration
Configuration Parameters:
Test Connection to Content Repository
OK
Use JAAS for security
Test
Help
< Back
Next >
Finish
Cancel

Repository Type: File System

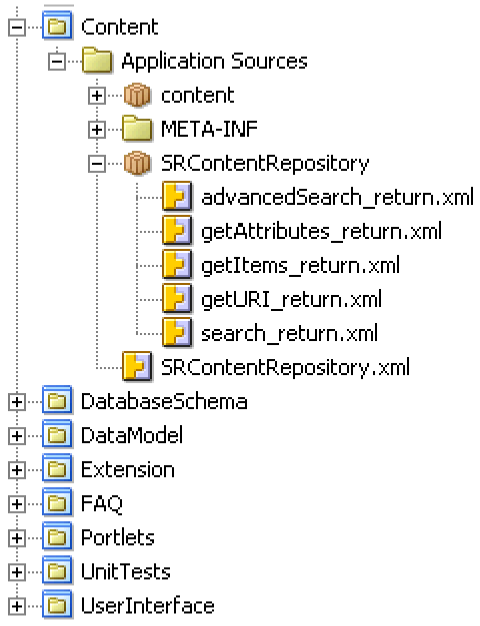
Parameter	Value
Base Path	C:\srdemo\SRContentRepository

Success!

8. Click **OK** to close the message box, and then click **Finish**.

Your Applications Navigator should show new entries under `Content\Application Sources`, as shown in Figure 4–6.

Figure 4–6   Files for the New Data Control

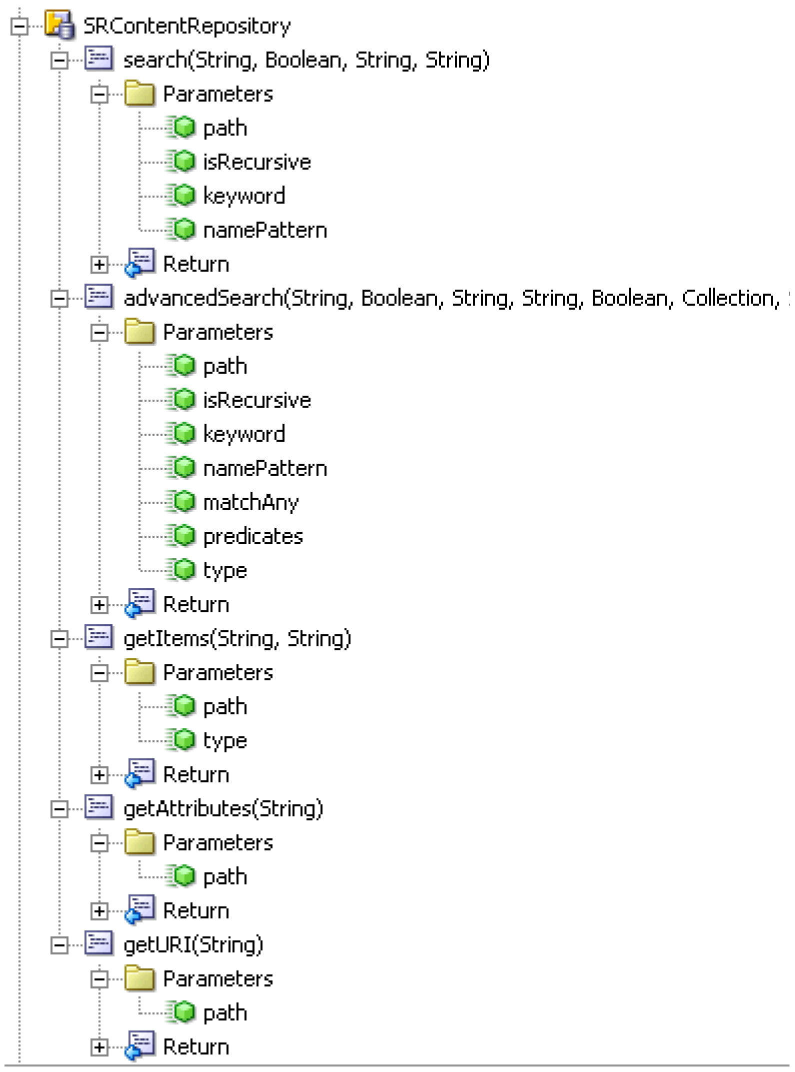


```
Content
  Application Sources
    content
    META-INF
    SRContentRepository
      advancedSearch_return.xml
      getAttributes_return.xml
      getItems_return.xml
      getURI_return.xml
      search_return.xml
    SRContentRepository.xml
DatabaseSchema
DataModel
Extension
FAQ
Portlets
UnitTests
UserInterface
```

9. To display the methods of the SRContentRepository data control, select **Data Control Palette** from the View menu.

Under SRContentRepository, you should see a hierarchical list of methods, parameters, and operators for the new data control, as shown in Figure 4–7.

Figure 4–7 Data Control Palette - SRContentRepository



```
SRContentRepository
    search(String, Boolean, String, String)
        Parameters
            path
            isRecursive
            keyword
            namePattern
        Return
    advancedSearch(String, Boolean, String, String, Boolean, Collection,
        Parameters
            path
            isRecursive
            keyword
            namePattern
            matchAny
            predicates
            type
        Return
    getItems(String, String)
        Parameters
            path
            type
        Return
    getAttributes(String)
        Parameters
            path
        Return
    getURI(String)
        Parameters
            path
        Return
```

Summary

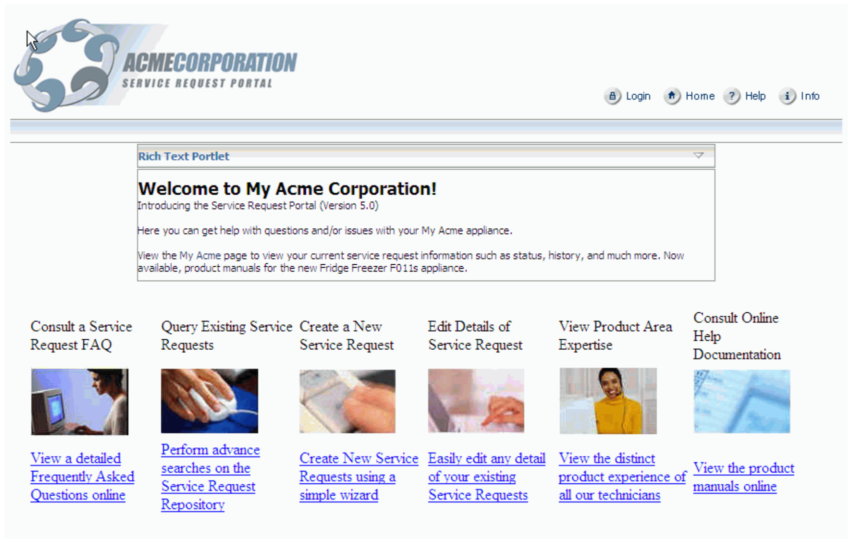
In this chapter, you learned to configure your content repository from a content directory located on your local drive. You will access the content repository in Chapter 5, "Creating a Public Welcome Page".

5

Creating a Public Welcome Page

This chapter shows you how to create a welcome page in the SRDemo application and customize it for your users. You will add a rich text component that displays announcement text to the end user and display a file in place from the sample content repository you set up in Chapter 4, "Setting Up Your Content Repository". You will then make the welcome page public so that any user can view it, as well as add a Login link so that users can enter their user ID and password to view information in the application specific to them. The welcome page you create will look like the Figure 5–1.

Figure 5–1 Overview of the Welcome Page



Welcome to My Acme Corporation!
Introducing the Service Request Portal (Version 5.0)

Here you can get help with questions and/or issues with your My Acme appliance.

View the My Acme page to view your current service request information such as status, history, and much more. Now available, product manuals for the new Fridge Freezer F011s appliance.

Consult a Service Request FAQ	Query Existing Service Requests	Create a New Service Request	Edit Details of Service Request	View Product Area Expertise	Consult Online Help Documentation
View a detailed Frequently Asked Questions online	Perform advance searches on the Service Request Repository	Create New Service Requests using a simple wizard	Easily edit any detail of your existing Service Requests	View the distinct product experience of all our technicians	View the product manuals online

This chapter includes the following sections:

- Step 1: Adding a Rich Text Component
- Step 2: Displaying a File In Place
- Step 3: Adding Global Navigation Links
- Step 4: Making the Welcome Page Public

Step 1: Adding a Rich Text Component

The steps in this section show you how to add a rich text component to the existing welcome page in the application. This rich text component will display announcements to the public user. The manager of the company, who has site administration access, can update this announcement at runtime.

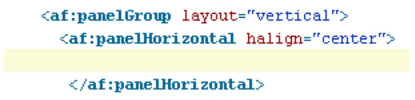
To add a rich text component:

1. From the **SRDemoSample_Starter** application, open the Design view of the `SRWelcome.jspx` page, located under **UserInterface**, **Web Content**, **app**.
2. Locate the image `acmecenter.jpg`.
3. Click the **Source** tab to view the source code of the page.
4. Delete the image from the page by removing the following code:

   ```
   <af:objectImage height="376"
           width="800"
           source="../images/acmecenter.jpg"/>
   ```

   This snippet of source code should now look like Figure 5–2.

Figure 5–2   panelGroup with the Acmecenter.jpg Deleted



```
<af:panelGroup layout="vertical">
  <af:panelHorizontal halign="center">

   </af:panelHorizontal>
```

   You will now add the Rich Text portlet at this location.

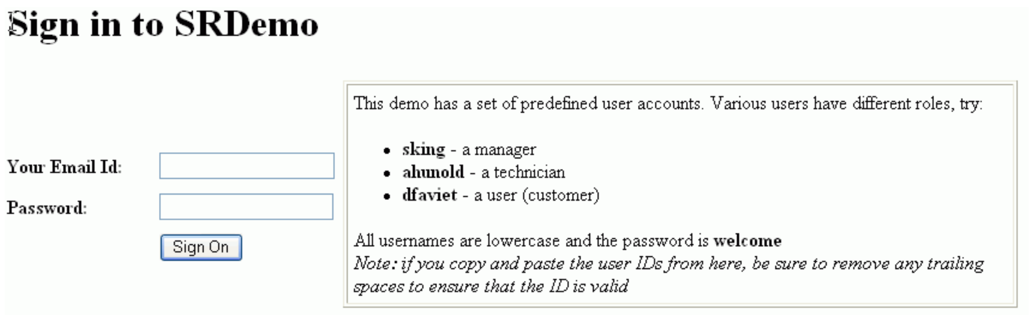
5. Before you can add the Rich Text portlet, you must first register it. However, to find the registration URL for the Rich Text Portlet Producer, do the following first:
   a. Start the preconfigured OC4J.
   b. If the readme page does not open by default, from the **Help** menu, select **WebCenter Preconfigured OC4J Readme**.
   c. On the readme page, click the **index page** link in the Index Page section.
   d. On the WebCenter Preconfigured OC4J index page, click **Rich Text Portlet Producer** under WSRP Portlet Producers.
   e. Click **WSRP v2 WSDL**.
   f. Save the URL from the address bar of the browser. Use this URL during the producer registration.
6. To register the Rich Text portlet producer, right-click the **UserInterface** project and choose **New**.
7. In the New Gallery dialog box, filter by All Technologies, then choose **Web Tier**, **Portlets**, **WSRP Producer Registration**.
8. In the Name field, enter `RichTextPortletProducer`, then click **Next**.
9. In the URL Endpoint field, enter the URL for the WSDL of the WSRP v2 Producer in your preconfigured OC4J. This is the URL you saved in step 5.
10. Click **Next**, then click **Finish**.
11. In the Structure pane, click to select the `panelHorizontal` component that originally contained the image.
12. From the Component Palette, choose **RichTextPortletProducer**, then select **Rich Text Portlet** from the list. The new portlet displays within the `panelHorizontal` component.

13. Select the **Rich Text Portlet** and open the Property Inspector.
14. Set the **AllModesSharedScreen** property to `true`.
15. Set the **Width** to `600` pixels.
16. Right-click **SRWelcome.jspx** and go to the page definition.
17. Edit authorization and grant **Customize** privilege to *manager*.

    You must do this as managers should be able to customize the content of the Rich Text portlet on the page.
18. Save the page.
19. In the Applications Navigator, right-click **SRWelcome.jspx** and choose **Run**.

    Due to a standard J2EE security constraint on all JSP and JSPX files, you will automatically be redirected to the default SRDemo login page, which is a simple JSP page that contains static HTML. You can log on as three different users: `ahunold`, `sking`, and `dfaviet` as shown in Figure 5–3.

Figure 5–3 The SRDemo Login Page



Sign in to SRDemo

Your Email Id:

Password:

Sign On

This demo has a set of predefined user accounts. Various users have different roles, try:

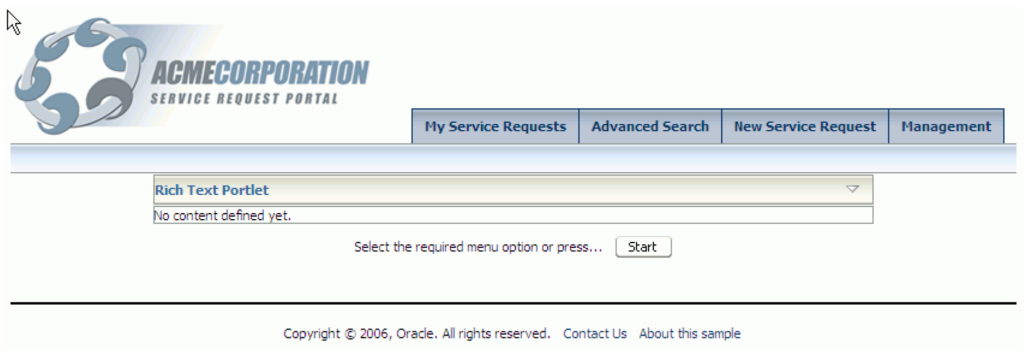
- **sking** - a manager
- **ahunold** - a technician
- **dfaviet** - a user (customer)

All usernames are lowercase and the password is **welcome**
Note: if you copy and paste the user IDs from here, be sure to remove any trailing spaces to ensure that the ID is valid

If you log in as `sking`, then you will see all the tabs, including the management tab shown on the far right in the following figure.

The portlet at runtime looks like Figure 5–4.

Figure 5–4 Tabs Displayed When Logged In As Manager



No content defined yet.

Select the required menu option or press...

Copyright © 2006, Oracle. All rights reserved.

If you log in as `ahunold`, you will not see the management tab, because `ahunold` is not a manager.

20. In your browser, click the arrow on the Rich Text portlet, then choose **Customize** to customize the text of the portlet.
21. Customize the portlet to include the following text:

```
Welcome to My Acme Corporation!

Introducing the Service Request Portal (Version 5.0)

Here you can get help with questions and/or issues with your My Acme appliance.

View the My Acme page to view your current service request information such as
status, history, and much more. Now available, product manuals for the new
Fridge Freezer F011s appliance.
```

To format this text:

Note: To avoid issues with text formatting, first copy this text to Notepad and then copy it from there to your portlet.

---

a. Select the first sentence and choose a larger font, then click the Bold icon.

b. To create a hyperlink in the text, select the **My Acme text** (underlined in the sample text), then click the **Hyperlink** icon in the Rich Text Editor. Enter the URL you selected into the field, for example:

   `http://localhost:port/SRDemo/faces/app/SRWelcome.jspx`

Step 2: Displaying a File In Place

Users have many options on how to display content by using content management systems. While a common way is to display the content as links, developers may also want to display the content of a text or HTML file directly within the application, due to formatting or for other reasons. Using JCR, you can display the file in place and still manage the content from within the content management system.

The steps in this section show you how to display a file from your content repository on an existing page. This lets you edit the page without changing the application that is rendering the page.

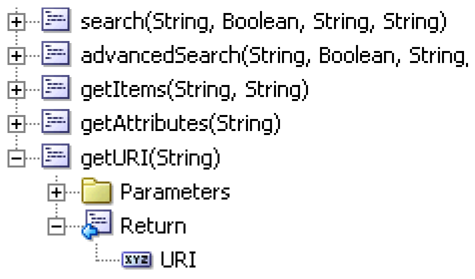
To display a file in place:

1. If it is not already open, open the existing `SRWelcome.jspx page` from **SRDemoStarter**, **UserInterface**, **Web Content**, **app**.
2. From the ADF Faces Core library, select a `panelHorizontal` and drag it on to the `panelPage`, below the `panelHorizontal` that contains the Rich Text portlet.
3. Set the panelHorizontal **halign** to `center`.
4. In the Data Control Palette, expand **SRContentRepository**. You created this data control earlier in Chapter 4, "Step 3: Configuring a JCR Data Control".

   Tip: If you cannot see the Data Control Palette, select Data Control Palette from the View menu.

5. Under `getURI(String)`, expand the **Return** node and drag **URI** (Figure 5–5) into the panelHorizontal.

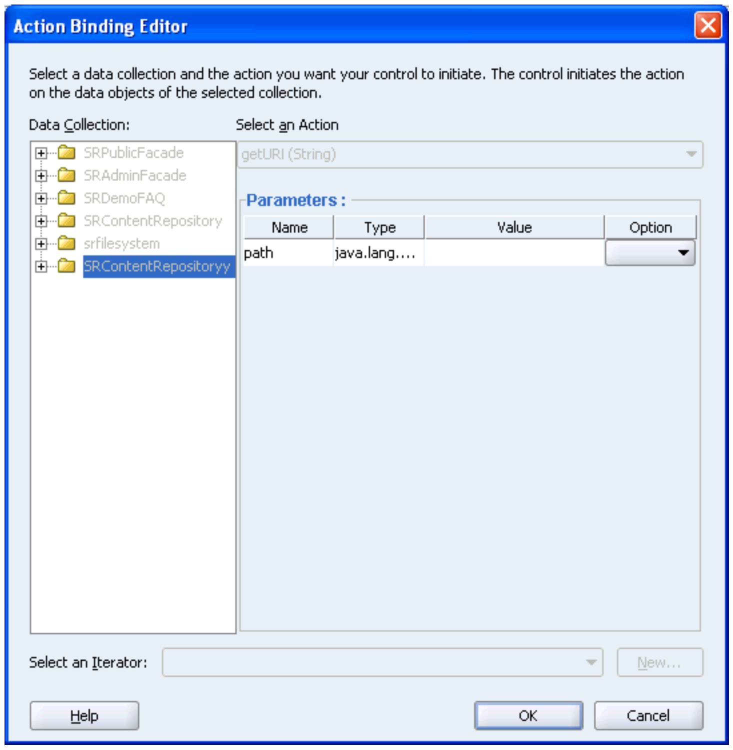
Figure 5–5 getURI in the Data Control Palette


search(String, Boolean, String, String)
advancedSearch(String, Boolean, String,
getItems(String, String)
getAttributes(String)
getURI(String)
Parameters
Return
URI

6. From the Create menu, select **Text** and **ADF Output Text**.

   This displays the Action Binding Editor, as shown in Figure 5–6.

Figure 5–6 Action Binding Editor



Select a data collection and the action you want your control to initiate. The control initiates the action on the data objects of the selected collection.

Name	Type	Value	Option
path	java.lang....		

7. In the Value field for the path parameter, enter `/welcome.html`.
8. Click **OK**.
9. View the source of the page and add the code shown in Example 5–1 below the `OutputText`:

Example 5–1 getURI Sample Code

```
<f:verbatim>
   <iframe height="450" width="850"
src="${pageContext.request.contextPath}${bindings['getURI_returnURI'].inputValue}"
frameborder="0">
</iframe>
</f:verbatim>
```

10. Save the file.
11. Run `SRWelcome.jspx`.

Enable Public Users to Execute the getURI Method

Since the welcome page is a public page, you must define that the `getURI` method (used to show the file from the content repository) can be executed by public users.

To change the access privileges:

1. Open the page definition.
2. Under `bindings` in the structure pane, right-click **getURI1**.
3. From the pop-up menu, select **Edit Authorization**.

   Refer to the chapter titled "Securing Your WebCenter Application" in *Oracle WebCenter Framework Developer's Guide* for more information about setting user privileges.
4. In the dialog box, select **invoke** against the *Anyone* role for the method `oracle.SRDemo.......getURI1_return`.

You should see the welcome page displayed as well as a URL which is produced by the `OutputText`. Go back to Oracle JDeveloper and delete the `OutputText` from the structure pane.

Step 3: Adding Global Navigation Links

You will have noticed that running `SRWelcome.jspx` redirects you to the default SRDemo application login page. In the initial starter version of the SRDemo, authorization within the application is implemented through J2EE security role membership.

J2EE security secures a path based on roles. Constraints are defined for specific URL patterns and these are mapped to constraints that are in turn mapped to roles. For example, the URL pattern `faces/app/*` can be mapped to all available roles so that everybody in those specific roles can access all the pages in the application. You can configure constraints for your application to recognize in the `web.xml` file. For example, you can configure a constraint that maps the manager role to the URL pattern `faces/app/manager/*` as shown in Example 5–2 (new constraint shown in **bold** text):

Example 5–2 Example of the Security Constraints in the web.xml file

```
<!-- Security Constraints ================================================== -->
 <security-constraint>
   <web-resource-collection>
     <web-resource-name>J2EE-Secured-Application</web-resource-name>
     <url-pattern>faces/app/*</url-pattern>
   </web-resource-collection>
   <auth-constraint>
      <role-name>user</role-name>
      <role-name>staff</role-name>
      <role-name>technician</role-name>
      <role-name>manager</role-name>
   </auth-constraint>
 </security-constraint>
<security-constraint>
   <web-resource-collection>
     <web-resource-name>J2EE-Secured-Application</web-resource-name>
     <url-pattern>faces/app/manager/*</url-pattern>
   </web-resource-collection>
   <auth-constraint>
      <role-name>manager</role-name>
   </auth-constraint>
 </security-constraint>
```

Refer to the chapter titled "Securing Your WebCenter Application" in *Oracle WebCenter Framework Developer's Guide* for more information about defining security constraints.

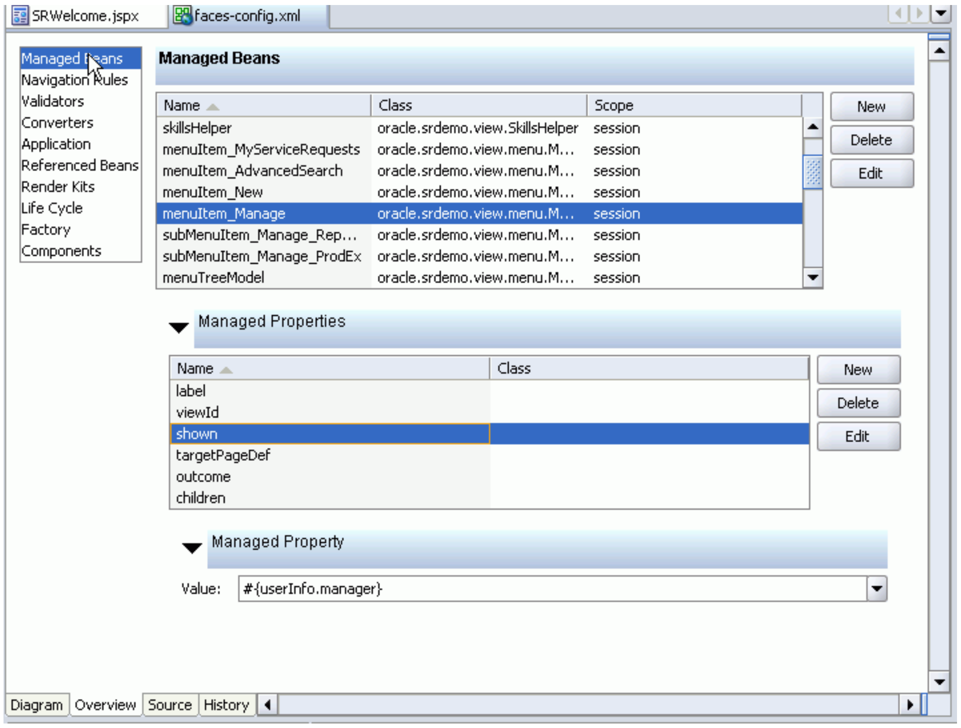
At runtime, a deployment descriptor maps the roles to the identity store. The policies, that is, the definition of which users belong to which roles and what they have access to, are specified in a JAZN resource providers, like JAZN-XML, which specifies its settings in the file `system-jazn-data.xml` or JAZN-LDAP, which can use Oracle Internet Directory. In the Oracle JDeveloper development environment, we use the `system-jazn-data.xml` file to store this policy information as shown in Example 5–3.

Example 5–3 Definition of the User and Role Mapping in the system-jazn-data.xml File

```
<role>
  <name>manager</name>
  <members>
   <member>
      <type>user</type>
      <name>sking</name>
   </member>
  </members>
</role>
```

Additionally, Expression Language can be used in the application to show and hide particular menu items like the management tab, which only shows up for managers. Figure 5–7 shows how the management tab is shown or hidden based on the membership of the Manager role.

Figure 5–7 Display of UI Elements Based on Membership



Managed Beans

Name	Class	Scope
skillsHelper	oracle.srdemo.view.SkillsHelper	session
menuItem_MyServiceRequests	oracle.srdemo.view.menu.M...	session
menuItem_AdvancedSearch	oracle.srdemo.view.menu.M...	session
menuItem_New	oracle.srdemo.view.menu.M...	session
menuItem_Manage	oracle.srdemo.view.menu.M...	session
subMenuItem_Manage_Rep...	oracle.srdemo.view.menu.M...	session
subMenuItem_Manage_ProdEx	oracle.srdemo.view.menu.M...	session
menuTreeModel	oracle.srdemo.view.menu.M...	session

Managed Properties

Name	Class
label	
viewId	
shown	
targetPageDef	
outcome	
children	

Managed Property

Value: #{userInfo.manager}

This internally maps to the `isManager` function in the `UserInfo` class, which determines membership of the named role as shown in Example 5–4.

Example 5–4 EL to Define What UI Elements Must Be Displayed

```
/**
  * Function designed to be used from EL for rendering UI Features based on
  * membership of the "manager" role.
  * @return boolean
  */
 public boolean isManager() {
        return (checkIsUserInRole("manager"));
 }
```

In the WebCenter version of the SRDemo, we want to implement a finer-grained security model, based on the ability to perform a specific action, rather than role membership.

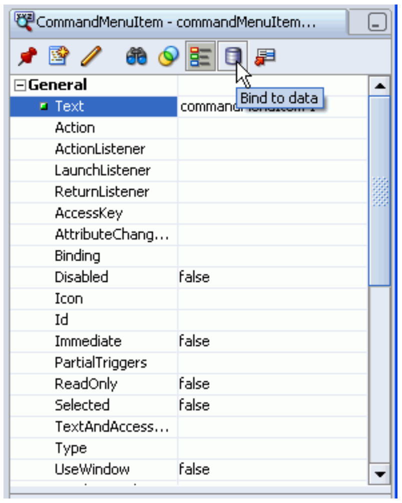
Unlike J2EE security, which is based on URLs, the use of JAAS-based ADF Security allows us to implement different levels of access for a given page. For example, View, Customize, and Edit.

To add ADF security-based global navigation links to the page, perform the following steps:

1. In the Applications Navigator, open `SRWelcome.jspx`.
2. In the Structure pane, navigate to **menuGlobal PanelPage facets**.
3. In the design view, drag a `menuButtons` component onto the `menuGlobal`.
4. Next, drag three `CommandMenuItems` into the `menuGlobal` facet on the top right corner of the page. You can also drop them into the appropriate node in the structure pane.

5. For the first `commandMenuItem`, select the **Text** property in the Property Inspector and add an expression to the text property by clicking the **Bind to Data** icon as shown in Figure 5–8. This displays the Bind to Data dialog box.

Figure 5–8 CommandMenuItem Properties



General	
Text	comman
Action	
ActionListener	
LaunchListener	
ReturnListener	
AccessKey	
AttributeChang...	
Binding	
Disabled	false
Icon	
Id	
Immediate	false
PartialTriggers	
ReadOnly	false
Selected	false
TextAndAccess...	
Type	
UseWindow	false

6. Under `JSP Objects`, open the `res` class and scroll down to the `srdemo.menu.home` entry. Select this and press the right arrow to return the required expression language, and click **OK**.
7. Set the **Action** property for the `commandMenuItem` to `GlobalWelcomeHome`.

   The icons for the various skins we are going to use are stored in individual skin images directories and we can use the skin bean (described in chapter 4) to return the directory structure that relates to the current skin which can be wrapped in EL
8. Set the **Icon** property for the first `commandMenuItem` as follows:

```
/skins/#{skinBean.currentSkin}/skin_images/home.gif
```

> **Note:** Some of the skins' icons contain text to indicate the action, while others require a separate text label. The skin bean has a function to define which skins require just an icon and which skins require icons plus text.

9. As the default skin already has labels in the icons, you must turn off the navigation buttons' text labels by setting the Text property of the commandMenuItem to the following:

```
text="#{skinBean.iconOnly ? null : res['srdemo.menu.home']}"
```

10. Repeat the same steps for the other two `commandMenuItems` and set the properties as follows:

    - For `commandMenuItem` two, select `GlobalHelp` for the **Action** property and `srdemo.menu.help` for the **Text** property.

      Set the **Icon** property for the second `commandMenuItem` as follows:

```
/skins/#{skinBean.currentSkin}/skin_images/help.gif
```

- For `commandMenuItem` three, select `GlobalAbout` for the **Action** property and `srdemo.menu.info` for the **Text** property.

  Set the **Icon** property for the third `commandMenuItem` as follows:

  ```
  /skins/#{skinBean.currentSkin}/skin_images/info.gif
  ```

In the Source view, the `commandMenuItem` definitions will now look like Example 5–5.

Example 5–5 Example Language Used for the Action of a Command Button

```
<af:menuButtons>
  <af:commandMenuItem text="#{skinBean.iconOnly ? null : res['srdemo.menu.home']}"
                      action="GlobalWelcomeHome"
                      icon="/skins/#{skinBean.currentSkin}/skin_images/home.gif"/>
  <af:commandMenuItem text="#{skinBean.iconOnly ? null : res['srdemo.menu.help']}"
                      action="GlobalHelp"
                      icon="/skins/#{skinBean.currentSkin}/skin_images/help.gif"/>
  <af:commandMenuItem text="#{skinBean.iconOnly ? null : res['srdemo.menu.info']}"
                      action="GlobalAbout"
                      icon="/skins/#{skinBean.currentSkin}/skin_images/info.gif"/>
</af:menuButtons>
```

11. In the structure pane, add a **goMenuItem** right before the first `commandMenuItem`.

  a. Select the **Text** property in the Property Inspector and add an expression to the text property by clicking the **Bind to Data** icon. Under JSF Managed Beans, scroll down to the authNLink entry. Select the **label** property and press the right arrow to return the required expression language, `text="#{authNLink.label }"`, which is then further updated.

  b. Click **OK**.

  Because some skins have text included in their icons, you must update the text property to allow for icons that contain a label by setting the property as follows:

  ```
  text="#{skinBean.iconOnly ? null : authNLink.label }"
  ```

  c. Set the Destination property to the **JSF Managed Beans**, **authNLink URL** as follows:

  ```
  destination="#{authNLink.URL}"
  ```

  d. Set the icon property to a context-sensitive skin icon as follows:

  ```
  /skins/#{skinBean.currentSkin}/skin_images/#{authNLink.icon}
  ```

AuthnLink is a managed bean that keeps track of the user's authenticated state and sets the login and logout link and associated icons automatically. See the chapter titled "Securing Your WebCenter Application" in the *Oracle WebCenter Framework Developer's Guide* for more information.

Note: The SRDemo version of the AuthnLink login component defines a boolean managed property (`returnHomeOnLogout`) that allows the developer to specify that the user should be returned to the application's home page (specified in the authnLink's `homePage managed` property) rather than staying on the current page when they log out of the application.

---

12. To display the name of the user who has logged in, open the Component Palette and select **JSF HTML** and then select **OutputFormat**. Drag it on the top of `infoUser` in the Structure pane.
13. Bind the Value property to `srdemo.connectedUser` under `JSP Objects/res`.
14. Bind the rendered property to `#{authNLink.authenticated}` under `JSF Managed Beans` (the value is a boolean that returns `true` or `false` based on the authenticated status).
15. In the Component Palette, select **JSF Core** and then select a **Param** component. Drag it to the Structure pane on top of the `outputFormat` object and set the following properties:
    - Name: `currentUserparam`
    - Value: `#{userInfo.userName}` (exposing the current user) (under JSF Managed Beans/userInfo)
    - Set escape: `false`

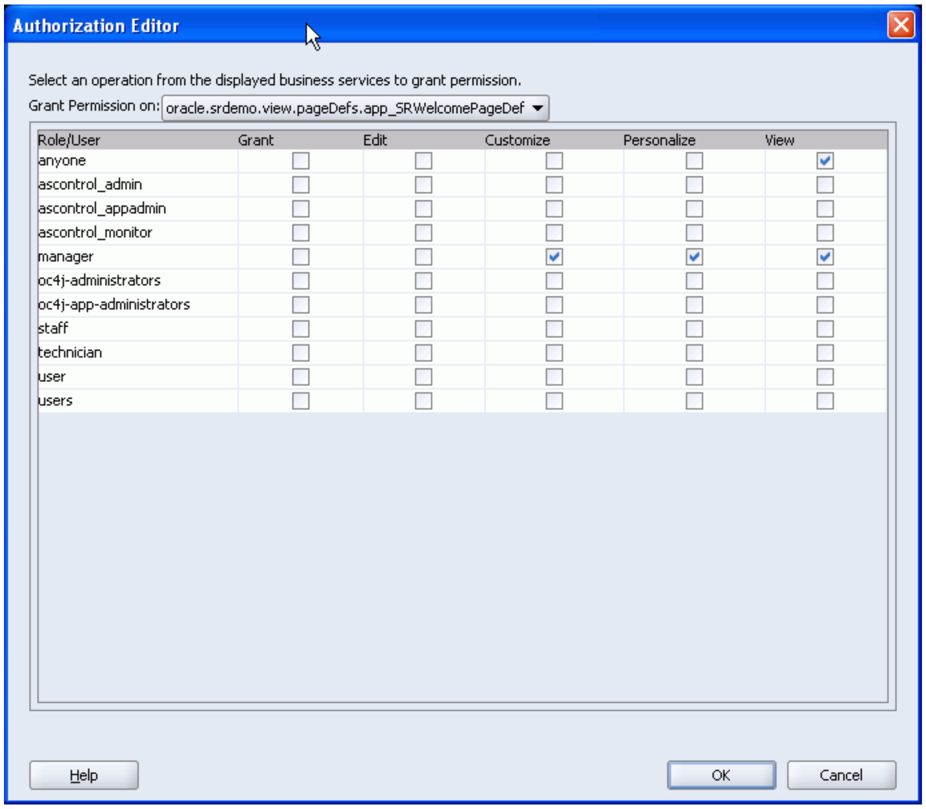
Note: The `AuthnLink.currentUser` property also returns the name of the current authenticated user and could be used to display the logged in user. In this case, however, we are using the userInfo bean for backward compatibility with the previous version of the SRDemo application.

Step 4: Making the Welcome Page Public

To make the welcome page ultimately publicly accessible (after ADF Security has been configured), you must define an access policy which states that the page will be viewable by all users, whether they are authenticated or anonymous (that is, available to anyone) To do this, perform the following steps:

1. Right-click **SRWelcome.jspx** and go to the page definition.
2. Edit authorization and grant **View** privilege to *anyone*, as shown in Figure 5–9.

Figure 5–9 Authorization Editor



Select an operation from the displayed business services to grant permission.

Grant Permission on: oracle.srdemo.view.pageDefs.app_SRWelcomePageDef

Role/User	Grant	Edit	Customize	Personalize	View
anyone	☐	☐	☐	☐	✓
ascontrol_admin	☐	☐	☐	☐	☐
ascontrol_appadmin	☐	☐	☐	☐	☐
ascontrol_monitor	☐	☐	☐	☐	☐
manager	☐	☐	✓	✓	✓
oc4j-administrators	☐	☐	☐	☐	☐
oc4j-app-administrators	☐	☐	☐	☐	☐
staff	☐	☐	☐	☐	☐
technician	☐	☐	☐	☐	☐
user	☐	☐	☐	☐	☐
users	☐	☐	☐	☐	☐

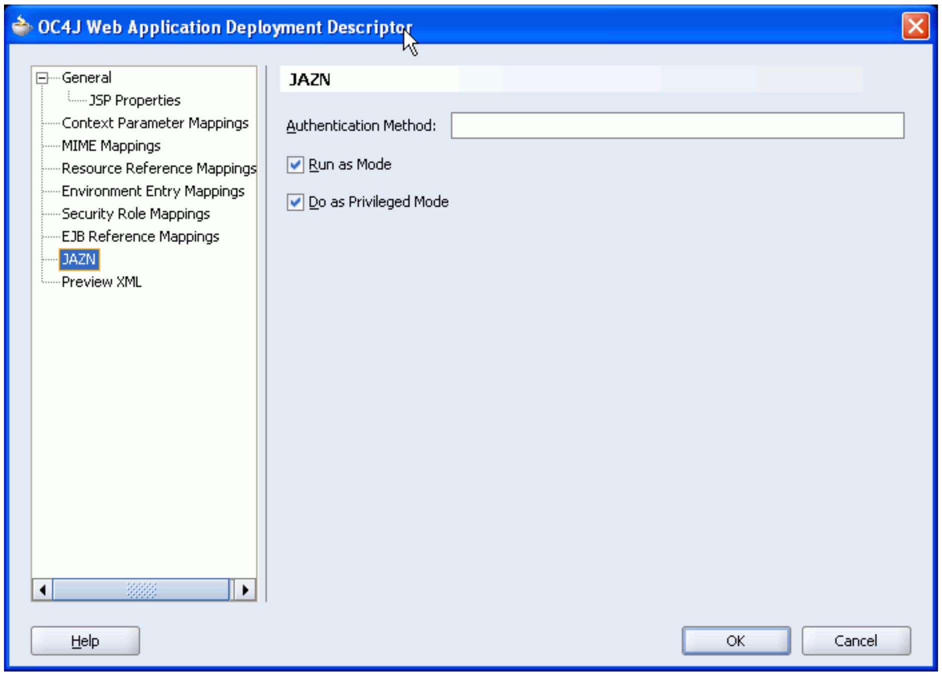
This will add the view privilege to the *anyone* role in the `system-jazn-data.xml` file. In an application protected by ADF security, (This will be turned on in the next chapter) each user is automatically made a member of this pseudo role (the anyone role principal is automatically added to the user's subject) Therefore, a public page is a special type of secured page that is available to everyone. This is different from J2EE security, in which a public page is defined by the simple lack of security constraints against that page. The ADF security model therefore differentiates between a page secured for public access and the absence of a secured implementation.

Note: We will run the page after we build the login page and activate ADF Security in the next chapter.

3. In the Applications Navigator, under the `Web Content/WEB-INF` folder, right-click the `orion-web.xml` file and choose **Properties**. The OC4J Web Application Deployment Descriptor dialog box is displayed.

4. Make sure the **Run as Mode** and **Do as Privileged Mode** options under **JAZN** are already selected, as shown in Figure 5–10.

Figure 5–10 orion-web.xml Properties


OC4J Web Application Deployment Descriptor
General
JSP Properties
Context Parameter Mappings
MIME Mappings
Resource Reference Mappings
Environment Entry Mappings
Security Role Mappings
EJB Reference Mappings
JAZN
Preview XML
Help
OK
Cancel

JAZN

Authentication Method:

[✓] Run as Mode

[✓] Do as Privileged Mode

5. Click **OK**.

Summary

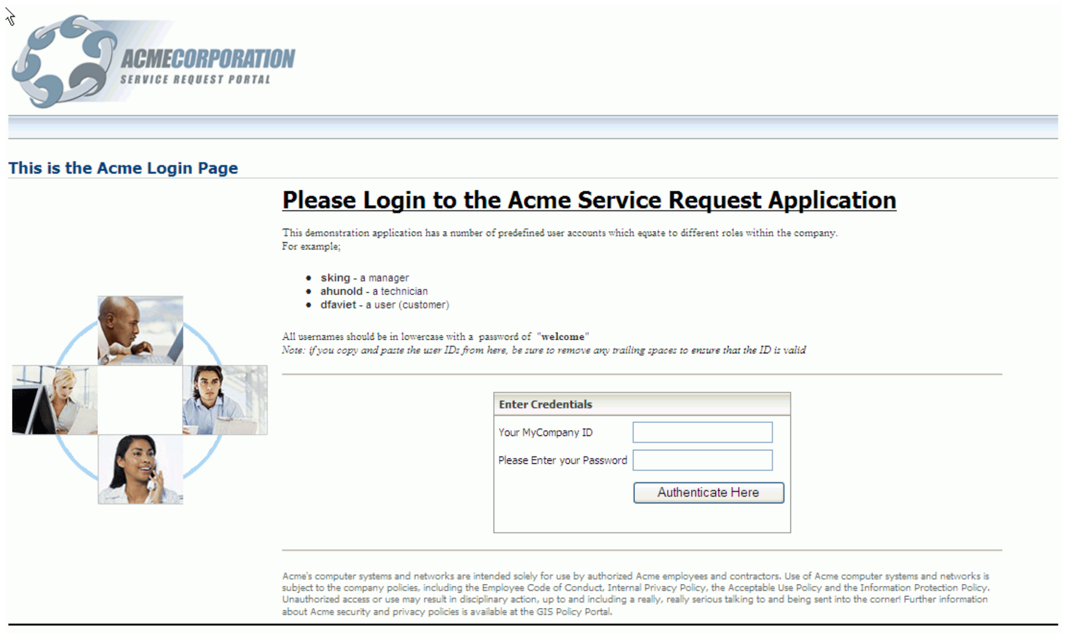
In this chapter, you learned how to create a welcome page, add portlets to it, and add content from a content repository. You also learned to make the page publicly available and add a login link for users to access information using their credentials.

6

Creating a Login Page

In this chapter you will create the new login page that users are redirected to for authentication. WebCenter applications typically have a notion of public pages and allow for explicit as well as implicit authentication. This means that users can log into the application by clicking the login link before they navigate to secured content (explicit), or they can navigate to a secured page, which will redirect them to the login page for the application (implicit). See "Securing your WebCenter Application" in the *Oracle WebCenter Framework Developer's Guide* for more information about implicit and explicit authentication. Figure 6–1 shows a sample login page you are going to build in this chapter.

Figure 6–1 Login Page From the SRDemo Application



This is the Acme Login Page

Please Login to the Acme Service Request Application

This demonstration application has a number of predefined user accounts which equate to different roles within the company.
For example;

- **sking** - a manager
- **ahunold** - a technician
- **dfaviet** - a user (customer)

All usernames should be in lowercase with a password of "**welcome**"
Note: if you copy and paste the user IDs from here, be sure to remove any trailing spaces to ensure that the ID is valid

Enter Credentials

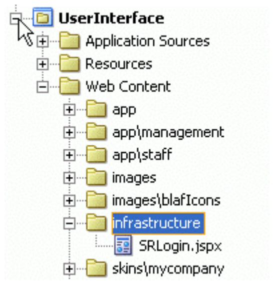
Your MyCompany ID	
Please Enter your Password	

Authenticate Here

Acme's computer systems and networks are intended solely for use by authorized Acme employees and contractors. Use of Acme computer systems and networks is subject to the company policies, including the Employee Code of Conduct, Internal Privacy Policy, the Acceptable Use Policy and the Information Protection Policy. Unauthorized access or use may result in disciplinary action, up to and including a really, really serious talking to and being sent into the corner! Further information about Acme security and privacy policies is available at the GIS Policy Portal.

The `SRDemoSample_Starter` application already contains a login page, `SRLogin.jspx`, which is located as shown in Figure 6–2.

Figure 6–2 Original SRDemo Login Page



Since the login page uses a login form, you do not typically use an ADF Faces page for a login page, because the ADF Faces lifecycle will remap the login form's submit action and its form fields to internal references, which would break the login functionality. However, the login page is an important part of the application. It can contain customizable information such as portlets and data controls and it can also benefit from *skinning* features that are available to ADF Faces pages. Alternatively, you may also want to implement a login portlet. In this chapter you will learn how you can build an ADF Faces-based login page for the SRDemo application. This login page's form also uses the J2EE security container login method, `j_security_check`. This chapter includes the following sections:

- Step 1: Creating a Login Page
- Step 2: Creating a Login Error Page
- Step 3: Adding a Rich Text Portlet to the Login Page
- Step 4: Editing Authorization for the Login Page
- Step 5: Configuring the Application to Use ADF Security and the Login Page
- Step 6: Updating Web.xml to Call the New Login Page
- Step 7: Running the Application

Step 1: Creating a Login Page

To create the ADF Faces-based login page, perform the following steps:

1. In the Applications Navigator, under the UserInterface project, expand the **Web Content** folder if necessary.
2. Right-click the **infrastructure** folder (shown in Figure 6–2) and select **New**.
3. In the New Gallery dialog box, expand the **Web Tier** node.
4. Select **JSF**.
5. In the Items list, select **JSF JSP**.
6. Click **OK** to display the Create JSF JSP dialog box.
7. If you are on the Welcome page of the wizard, click **Next** to display the JSP File page.
8. In the **File Name** field, enter `SRLoginADF.jspx`.
9. Select **JSP Document (*.jspx)**.
10. Click **Next** to display the Component Binding page.
11. Select **Automatically Expose UI Components in a New Managed Bean**.

12. Click **Next** to display the Tag Libraries page and select **ALL Libraries**.
13. Make sure that the following libraries are listed in Selected Libraries:
    - JSF Core
    - JSF HTML
    - ADF Faces Components
    - ADF Faces HTML
    - ADF Portlet Components
    - Customizable Components Core
14. Click **Finish** to create the page.
15. Save the page.
16. In the Source View of the `SRLoginADF.jspx` page, copy the code shown in Example 6–1 and paste just before the `f:view` tag.

Example 6–1 Reference to the SRDemo Resource Bundle

```
<!-- Resource Bundle for Translatable Strings within Application =========== -->
<f:loadBundle basename="oracle.srdemo.view.resources.UIResources" var="res"/>
```

17. Save the file.
18. Switch to Design View.
19. From the Component Palette, select **ADF Faces Core**.
20. Select a **PanelPage** component and drag it on to the `body` element in the Structure pane. When prompted, do *not* add a form, because we will add our own form later and delete the default form element.
21. Change the `Title` property, by binding it to a string from the resource bundle that ships with the application. In the Property Inspector, click **Title**.
22. Click the **Bind to Data** icon (the database icon at the top of the Property Inspector). This displays the Bind to Data dialog box.
23. Under **JSP objects**, select **res** and find the `srlogin.pageTitle` variable.
24. Move it to the Expression field and click **OK**.
25. Add some facets in the page title key, `srlogin.pageTitle`. To do so, find the existing branding facet code shown in Example 6–2.

Example 6–2 The Original Branding Facet

```
<f:facet name="branding"/>
```

26. Replace the branding image facet code with the facet code shown in Example 6–3. This will set the logo image within the branding facet to reflect the appropriate company branding logo for the chosen skin.

Example 6–3 The New Branding Facet

```
<!-- Site Branding Section (Top Left of Page) ============================== -->
      <f:facet name="branding">
            <af:objectImage source="/skins/#{skinBean.currentSkin}/skin_
images/SRBranding.gif"/>
      </f:facet>
```

27. Next, add the copyright facet to the new login page. In the `SRLoginADF.jspx` page, find the existing copyright facet code shown in Example 6–4.

Example 6–4 The original Copyright facet

```
<f:facet name="appCopyright"/>
```

28. Replace this copyright facet with the facet shown in Example 6–5. This will load the branding image from the skin bean, which ensures that it can be changed at run time, when the skin is changed.

Example 6–5 The Copyright Facet

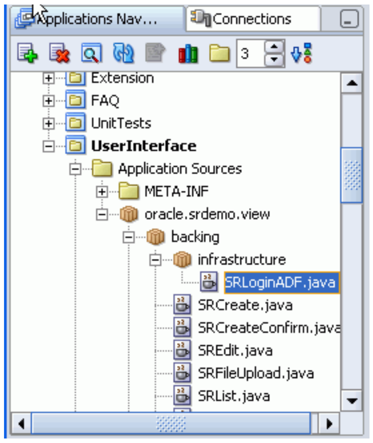
```
<!-- Copyright Message -->
     <f:facet name="appCopyright">
       <h:panelGroup>
         <af:outputText value="#{res['srdemo.copyright']}"/>
         <af:objectSpacer width="10" height="10"/>
       </h:panelGroup>
     </f:facet>
```

29. Now that you have created the login page as an ADF Faces page, you cannot just add the login form using form elements from the Component Palette. This would cause the form elements to be serialized and remapped at runtime by the ADF Faces lifecycle. Instead, we will design the page such that the login component is independent from the form surrounding the active JSF components (Those that submit back to the Faces lifecycle). While this could be achieved by using verbatim tags inside the JSF page, it greatly increases the complexity and limits the flexibility available. To allow the login component to be treated as a single element (rather than a series of escaped HTML tags), which can be moved around the page and easily skinned, you can separate the HTML form from the User Interface and inject it dynamically at runtime. This can be achieved by including it in the backing bean and exposing the returned HTML in an OutputText object that is rendered in the User Interface. To do this, perform the following steps:

Note: The code is available in the `SRLoginADF_Backing_Code.txt` file located in the starter ZIP file (in the root directory).

a. Open the Backing Bean, `SRLoginADF.java`, located in the `backing/infrastructure` folder, as shown in Figure 6–3, to add the HTML injection code to the Backing Bean so that it can be pulled into the page at runtime. The basis for this code is to generate the appropriate HTML directly in the page. As such, the various getter methods return the appropriately marked up HTML as a simple string. The advantage of using this dynamically generated HTML is the ability to include references to the skin information. The integration of the HTML into the Faces page is simplified by removing the need for verbatim tags and CDATA escapes within the page.

Figure 6–3   Location of SRLoginADF.java in the Applications Navigator



b. Now add the Java code to the file. To do so, add the import statements shown in Example 6–6.

Example 6–6   Import Statement

```
import javax.faces.application.FacesMessage;
import javax.faces.context.FacesContext;
import javax.servlet.http.HttpSession;
import oracle.adf.share.ADFContext;
import oracle.adf.view.faces.component.core.output.CoreMessages;
import oracle.srdemo.view.util.JSFUtils;
```

c. Add Backing Bean properties to top of `SRLoginADF.java`, as shown in Example 6–7. These getter methods for these properties will later be referenced in Expression Language in the login page.

Example 6–7   Backing Bean Properties

```
private String  _loginStyleBlock   = null;
private String  _loginScriptBlock  = null;
private String  _loginFormBlock    = null;
private boolean _validLoginAttempt = true;
```

Table 6–1 describes these properties.

Table 6–1   Backing Bean Properties in the SRLoginADF.java File

Variable	Description
loginStyleBlock	Contains the HTML for defining a CSS class
loginScriptBlock	Contains the HTML for the required JavaScript used by the login form
loginFormBlock	Contains the HTML for the login form itself
validLoginAttempt	A boolean value that indicates whether the current user is able to perform another login attempt or not. This is set to `false` if the user exceeds the number of valid login attempts.

d. Insert the `getloginStyleBlock` getter method in the Backing Bean. See Example 6–8.

This generates a simple stylesheet that defines flashing text that is used by the login form to indicate that the authentication process is taking place. The HTML that is generated by this method is invisible, and as such, the output string can be rendered outside of the body of the page.

Example 6–8 getloginStyleBlock Getter Methods

```
/**
  * ============================================================================
  * Function to generate the CSS style class referenced in Login Form
  * ============================================================================
  * @return
  */
public String getloginStyleBlock()
{
  _loginStyleBlock = "\n\n"
                   + "<!--
========= CSS Style Block Generated in Backing Bean ========= -->\n"
                   + "<style type=\"text/css\">\n"
                   + ".VP_Blink {text-decoration: blink;}\n"
                   + "</style>\n"
                   + "<!--
 ============================================================= -->"
                   + "\n\n";
  return (_loginStyleBlock);
}
```

e. Insert the `getloginScriptBlock` getter method the Backing Bean. See Example 6–9.

This method generates the `<script>` block that contains the JavaScript functions that are used by the login form. These include field validation and form-submission actions. As the JavaScript is dynamically generated, it is possible to include the references to the current skin as well as appropriate strings from the resource bundle. For example, if the current skin is set to `myCompany`, then the following line sets the JavaScript alert message to be the value that is defined in the resource bundle key: `SRLoginADF.myCompany.jsAlert`.

```
String alertMsg    = JSFUtils.getStringFromBundle("SRLoginADF." +
currentSkin + ".jsAlert");
```

In a similar way, the method determines the correct icon to indicate that the authentication is in progress and generates the URL to that icon based on the Faces external context.

Example 6–9 getloginScriptBlock Getter Methods

```
/**
 * ============================================================================
 * Method getloginScriptBlock() to generate a JScript Submit form block.
 * This function implements the current skin and generates simple JavaScript
 * to validation the user input prior to the Authentication submit action.
 * ============================================================================
 * @return
 */
public String getloginScriptBlock()
{
 String currentSkin = (String)
 JSFUtils.getManagedBeanValue("skinBean.currentSkin");
 String alertMsg    = JSFUtils.getStringFromBundle("SRLoginADF."
                                               + currentSkin +
                                               ".jsAlert");
 String valPwdMsg   = JSFUtils.getStringFromBundle("SRLoginADF."
                                               + currentSkin +
                                               ".AuthMsg");
 String urlBaseRef  =
FacesContext.getCurrentInstance().getCurrentInstance().getExternalContext().getReq
uestContextPath();
 String processIcon = urlBaseRef + "/skins/" + currentSkin + "/skin_
images/process_animation.gif";

 _loginScriptBlock = "\n\n"
                   + "<!--
======== JavaScript Block Generated in Backing Bean ========= -->\n"
                   + "<SCRIPT language=\"JavaScript\" type=\"text/JavaScript\">\n"
                   + "(new Image(32,32)).src=\"" + processIcon +"\";\n"
                   + "function SubmitForm(AuthFrm){\n"
                   + "var alertMsg =\""   + alertMsg  + "\";\n"
                   + "var valPwdMsg = \"" + valPwdMsg + "\";\n\n"
                   + "if (((AuthFrm.j_username.value == null) || (AuthFrm.j_
username.value ==\"\")) ||\n"
                   + "    ((AuthFrm.j_password.value == null) || (AuthFrm.j_
password.value ==\"\")))\n"
                   + " {\n  alert(alertMsg);\n  return false;\n }\n else\n {\n"
                   + " var divTag = document.getElementById(\"vp\");\n"
                   + " divTag.innerHTML = '<table width=\"50\" border=\"0\"><tr>'
+\n"
                   + "                      '<td align=\"center\"><img src=\""
                                             + processIcon + "\" width=\"32\"
                                             height=\"32\"></td>' +\n"
                   + "                      '<td align=\"left\"><font size=\"-2\"
                                             class=\"VP_Blink\">' + valPwdMsg +
                                             '</font></td>' +\n"
                   + "                      '</tr></table>';\n"
                   + " return true;\n"
                   + " }\n}\n"
                   + "</SCRIPT>\n"
                   + "<!--
 ============================================================ -->"
                   + "\n\n";
return (_loginScriptBlock);
}
```

f. Insert the `getLoginFormBlock` getter method in the Backing Bean. See Example 6–10.

This method generates a string that defines the HTML login form that is rendered inside the page. As the HTML is dynamic it is able to include references to the current skin as well as keys in the resource bundle. Furthermore, as the login page is designed to be customized after login, the HTML is sensitive to the authenticated state of the user in that the HTML disabled property is dynamically set using the ADF security context's `isAuthenticated();` method.

In this case, if the login page is accessed once the user is authenticated, then the login page (while rendered) is deactivated to prevent subsequent login attempts. For example, this is the case when the page is accessed from the site administration page, as discussed in Chapter 9, "Building a Site Administration Page".

In the following example, you can see that the string variable `htmlDisable` is either `"disabled="true""`, or `null`, based on the ADF Security context's `isAuthenticated` state:

```
String htmlDisable
=(ADFContext.getCurrent().getSecurityContext().isAuthenticated() ?
"disabled=\"true\"" : null);
```

The following code snippet shows you how the `htmlDisable` string is subsequently used to activate or disable the form elements, such as input fields and Submit button:

```
<input type=\"text\" name=\"j_username\" "+ htmlDisable + "/></td>\n"
```

Example 6–10   getLoginFormBlock Getter Method

```
/**
 * ===========================================================================
 * Method getLoginFormBlock() generates the standard Credential Form used
 * by container security. The function implements the current skin values and
 * returns the required HTML which is subsequently output using an OutputText
 * object populated using EL. While this could also have been achieved by
 * directly using the Verbatim tags within the JSPX itself, this would require the
 * use of CDATA escapes and complicates the design of the page (the login form
 * is exposed through a faces component in this case.)
 *
 * Note as the Login Page contains customizable components, the HTML form elements
 * generated are disabled if the user is currently authenticated.
 * ==========================================================================
 * @return
 */

public String getLoginFormBlock()
{
  String currentSkin       = (String)
JSFUtils.getManagedBeanValue("skinBean.currentSkin");
  String userNameLabel     = JSFUtils.getStringFromBundle("SRLoginADF."
                                                     + currentSkin +
                                                     ".userName");
  String passwordLabel     = JSFUtils.getStringFromBundle("SRLoginADF."
                                                     + currentSkin +
                                                     ".password");
  String submitButtonLabel = JSFUtils.getStringFromBundle("SRLoginADF." +
                                                      currentSkin +
                                                     ".buttonLabel");
  String htmlDisable       =
(ADFContext.getCurrent().getSecurityContext().isAuthenticated() ?
"disabled=\"true\"" : null);

  _loginFormBlock = "\n\n"
                  + "<!--
======== Login Form Block Generated in Backing Bean ========= -->\n"
                  + "<form name=\"LoginForm\" id=\"LoginForm\" \n"
                  + " action=\"j_security_check\" method=\"POST\"
                      onSubmit=\"return SubmitForm(this)\" >\n"
                  + "<table cellspacing=\"5\" cellpadding=\"0\" border=\"0\"
                     width=\"50%\" bgcolor=\"#FFFFFF\" width=\"240\" >\n"
                  + " <tr>\n"
                  + "  <td nowrap>" + userNameLabel + "</td>\n"
                  + "  <td nowrap><input type=\"text\" name=\"j_username\"
                    "  + htmlDisable + "/></td>\n"
                  + " </tr>\n"
                  + " <tr>\n"
                  + "  <td nowrap>" + passwordLabel + "</td>\n"
                  + "  <td nowrap><input type=\"password\" name=\"j_password\"
                    " + htmlDisable + "/></td>\n"
                  + " </tr>\n"
                  + " <tr>\n"
                  + "  <td nowrap height=\"34\"><DIV id=\"vp\"></DIV></td>\n"
                  + "  <td nowrap>\n"
                  + "    <input type=\"submit\" value=\"" + submitButtonLabel +
                    "\" " + htmlDisable + "/></td>\n"
                  + "  </td>\n"
                  + " </tr>\n"
                  + "</table>\n"
                  + "</form>\n"
                  + "<!--
 ============================================================ -->"
                  + "\n\n" ;
  return ( _loginFormBlock);
}
```

g. Save the file.

30. From the Component Palette, select **ADF Faces Core**.
31. Select **PanelHorizontal** and drag it on to the `page` component.
32. Set the `Valign` property to `middle` in the Property Inspector.
33. Select **ObjectImage** from the ADF Faces Core Component Palette and drag it onto the `PanelHorizontal` component. Set the `source` attribute to the following value:

    ```
    /skins/#{skinBean.currentSkin}/skin_images/LoginSplash.gif
    ```

34. Select **ObjectSpacer** and drag it on to the page just below the `ObjectImage` component.
35. From the Component Palette, select **Customizable Components Core**.
36. Select **PanelCustomizable** and drag it below the `ObjectSpacer` element you just added in the Structure pane. When prompted, do *not* add a form, because we will add our own form later and delete the default form element.
37. Set the `Layout` property for the `PanelCustomizable` component to `vertical`.
38. From the Component Palette, select **ADF Faces Core**.
39. Now add the following to the `PanelCustomizable` component:
    - an objectSpacer
    - an objectSeparator
    - an objectSpacer
    - a panelHorizontal

      set the `Halign` property to `center`
    - an objectSpacer
    - an objectSeparator
    - an objectSpacer
40. Select **PanelBox** and drag it onto the `PanelHorizontal` component.

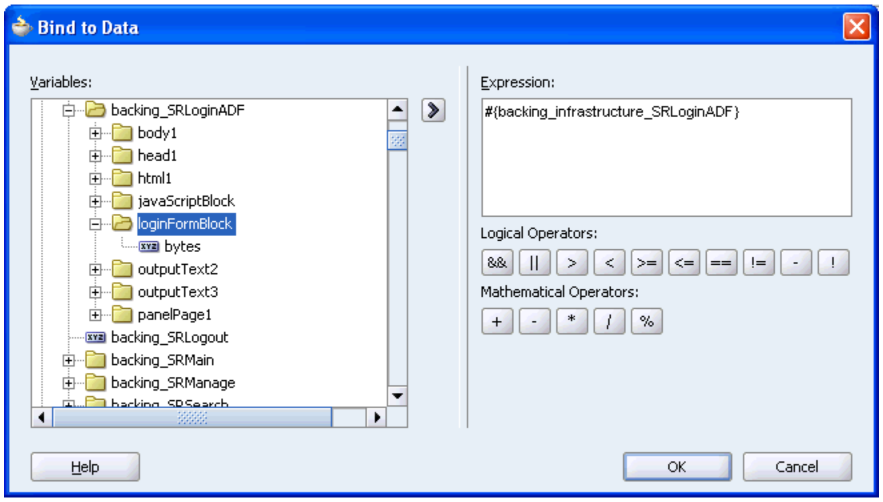
41. Set the `Text` property of the `PanelBox` to be the login component header by referencing the resource bundle, using expression language (EL). To do this, click the **Bind to Data** icon. This displays the Bind to Data dialog box.

42. Under **JSP objects**, select **res** and find the `SRLoginADF.credentialHeader` variable.

43. Select **OutputText** from the ADF Faces Core Component Palette and drag it onto the `PanelBox` component.

44. In the Property Inspector, bind the `Value` property of the `OutputText` component to data from a managed bean by selecting **JSF Managed Beans**, **backing_infrastructure_SRLoginADF**, and then **loginFormBlock** as shown in Figure 6–4.

Figure 6–4 Bind to Data

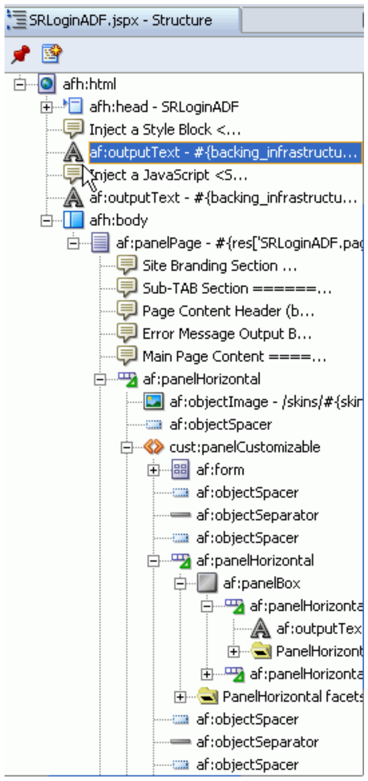


#{backing_infrastructure_SRLoginADF}

45. Set the `OutputText` component's `Escape` property to `false`.

46. Drag another **OutputText** component above the body tag in the Structure pane, just below the top of the page, as shown in Figure 6–5.

Figure 6–5 Structure Pane


SRLoginADF.jspx - Structure
afh:html
afh:head - SRLoginADF
Inject a Style Block <...
af:outputText - #{backing_infrastructu...
Inject a JavaScript <S...
af:outputText - #{backing_infrastructu...
afh:body
af:panelPage - #{res['SRLoginADF.pa
Site Branding Section ...
Sub-TAB Section ======...
Page Content Header (b...
Error Message Output B...
Main Page Content ====...
af:panelHorizontal
af:objectImage - /skins/#{skin
af:objectSpacer
cust:panelCustomizable
af:form
af:objectSpacer
af:objectSeparator
af:objectSpacer
af:panelHorizontal
af:panelBox
af:panelHorizonta
af:outputTex
PanelHorizont
af:panelHorizonta
PanelHorizontal facets
af:objectSpacer
af:objectSeparator
af:objectSpacer

47. Bind the `Value` property to data from the backing bean by selecting **JSF Managed Beans**, **backing_infrastructure_SRLoginADF**, and **loginStyleBlock** and set the `Escape` property to `false`.
48. Drag another **OutputText** component onto the Structure pane, just above the body tag.
49. Bind the `Value` property to data from the backing bean by selecting **JSF Managed Beans**, **backing_infrastructure_SRLoginADF**, and **loginScriptBlock**, and then set the `Escape` property to `false`.
50. In the `SRLoginADF.jspx` page, open the **panelPage** node in the Structure pane, then drag the **Messages** component onto the `messages` facet.
51. Save the file.

Step 2: Creating a Login Error Page

To create a login error page, perform the following steps:

1. In the Applications Navigator, under the UserInterface project, expand the **Web Content** node, if necessary.
2. Right-click the **infrastructure** folder (shown in Figure 6–2) and select **New**.
3. In the New Gallery dialog box, expand the **Web Tier** node.
4. Select **JSP**.
5. In the Items list, select **JSP**.
6. Click **OK** to display the Create JSP dialog box.

7. If you are on the Welcome page of the wizard, click **Next** to display the Create JSP page.
8. In the **File Name** field, enter `SRLoginErrorPage.jsp`.
9. On the Error Page Options page of the wizard, select **Do Not Use an Error Page to Handle Uncaught Exceptions in This File**.
10. On the Tag Libraries page, select **All Libraries**, and then select **JSTL Core 1.1** and move it to the Selected Libraries list.
11. Click **Finish**.
12. Add the JSTL code shown in Example 6–11 just before the `<html>` element to track the login attempts.

Example 6–11 JSTL Code to Track the Number of Login Attempts

```
<!-- ===========================================================================
 For the SRDemo Demo we will track the number of Login Attempts against the file
 based repository (In a deployed system this would be handled by the IdM system).
 This counter is then used in the backing bean to define if the login attempt
 is valid (referenced in EL as "loginAttemptValid").
 ======================================================================= -->
<c:set var="LoginErrorCount"
       scope="session"
       value="${sessionScope.LoginErrorCount == null ? 1
:sessionScope.LoginErrorCount + 1}"/>
<c:redirect url="/faces/infrastructure/SRLoginADF.jspx"/>
```

This JSTL code stores in a simple session variable the number of times the error page is called (due to invalid login) and redirects the user back to the original login page where the actual processing takes place.

13. Save the `SRLoginErrorPage.jsp` file.
14. Open the backing bean, `SRLoginADF.java` and add the `getMessages1()` method shown in Example 6–12.

The following two methods use the Faces external context to return the session variable that was set by the JSTL in the error page, to determine the number of times the user has attempted to log in.

The following code snippet shows you how to return the session variable based on the Faces context and the associated HTTP session. The returned object is converted to a local integer to evaluate whether the user's login attempt is valid or not. For example, a three-attempt limit can thus be enforced.

```
FacesContext facesContext = FacesContext.getCurrentInstance();
HttpSession  session =
(HttpSession)facesContext.getExternalContext().getSession(true);
Object loginAttempts  = session.getAttribute("LoginErrorCount");
Integer loginAttemptCount = (loginAttempts == null)? 0 :
Integer.parseInt(loginAttempts.toString());
```

Note: The `getMessages1` method is the default getter method created for the message component of the `PanelPage`. If your message component is not `messages1`, you must change the name of the method.

Having determined the current number of logging attempts, the `getMessages1` method sets the appropriate error string (returned from the resource bundle) for the Faces message component. In this case greater than one attempt indicates an invalid username and password combination and greater than three attempts indicate that the user has exceeded the legal limit.

Example 6–12 getMessage1() Method

```
/**
     * ==========================================================================
     * Function getMessages1() determines the current number of login failures and
     * sets the appropriate error message in the Faces Context
     * ==========================================================================
     * @return
     */
    public CoreMessages getMessages1()
    {
     if (!ADFContext.getCurrent().getSecurityContext().isAuthenticated())
     {
       String       currentSkin       = (String)
       JSFUtils.getManagedBeanValue("skinBean.currentSkin");
       FacesContext facesContext      = FacesContext.getCurrentInstance();
       HttpSession  session           = (HttpSession)
       facesContext.getExternalContext().getSession(true);
       Object       loginAttempts     = session.getAttribute("LoginErrorCount");
       Integer      loginAttemptCount = (loginAttempts == null)? 0 :
       Integer.parseInt(loginAttempts.toString());
       String       loginErrorMessage = null;
       if (loginAttemptCount >0)
       {
        if (loginAttemptCount >2) {
         loginErrorMessage = JSFUtils.getStringFromBundle(
                             "SRLoginADF." + currentSkin +
                             ".tooManyLoginAttempts");
        }
        else {
         loginErrorMessage = JSFUtils.getStringFromBundle(
                   "SRLoginADF." + currentSkin + ".invalidLogin");
        }
        FacesMessage fm =  new FacesMessage(
                  FacesMessage.SEVERITY_INFO,loginErrorMessage,null);
        facesContext.addMessage(null, fm );
       }
     }
     return messages1;
    }
```

15. Open the backing bean, `SRLoginADF.java` and add the `isValidLoginAttempt` method shown in Example 6–13.

    The boolean value returned by the `isValidLoginAttempt` method is referenced in the rendered property of the output text component which holds the login form HTML code. In this case, when the number of valid attempts is greater than three, the login form is not rendered on the page and the user is prevented from logging in again.

Example 6–13 isValidLoginAttempt Method

```
/**
 * ========================================================================
 * Method isValidLoginAttempt() evaluates the number of invalid login attempts
 * performed by the user and returns false if greater than 3. (used in EL to
 * show/hide the Login Page components.
 * ========================================================================
 * @return
 */
public boolean isValidLoginAttempt()
{
  _validLoginAttempt = true;
  if (!ADFContext.getCurrent().getSecurityContext().isAuthenticated()) {
     FacesContext facesContext      = FacesContext.getCurrentInstance();
     HttpSession  session           = (HttpSession)
     facesContext.getExternalContext().getSession(true);
     Object       loginAttempts     = session.getAttribute("LoginErrorCount");
     Integer      loginAttemptCount = (loginAttempts == null)? 0
    : Integer.parseInt(loginAttempts.toString());

     if (loginAttemptCount >2){ _validLoginAttempt = false; }
  }
  return (_validLoginAttempt);
}
```

16. Save the file.
17. Open `SRLoginADF.jspx`.
18. Drag an **ObjectImage** into the `PanelBox` below the `OutputText` object, and set the `source` property to the following value:

    ```
    source="/skins/#{skinBean.currentSkin}/skin_images/BadLoginAttempt.gif"
    ```

19. Bind the `rendered` property to the negated (opposite) value of the `isValidLoginAttempt()` method in the backing bean. Set the `rendered` property to the following value:

    ```
    #{!backing_infrastructure_SRLoginADF.validLoginAttempt}
    ```

    By doing this, the image will be rendered only if the login attempt is invalid.
20. Set the `rendered` property of the `OutputText` object to the value of the `isValidLoginAttempt()` method as follows:

    ```
    #{backing_infrastructure_SRLoginADF.validLoginAttempt}
    ```

    This ensures the login form is rendered only when it is valid for a user to try to login.

Step 3: Adding a Rich Text Portlet to the Login Page

In this section, you will add a Rich Text portlet to the login page. To do this, perform the following steps:

1. Make sure your Rich Text portlet has been registered before proceeding. Refer to "Step 1: Adding a Rich Text Component" in Chapter 5, "Creating a Public Welcome Page" for the details.
2. From the Component Palette, choose **RichTextPortletProducer**, then select **Rich Text** from the list. In the Structure pane, drag the **Rich Text Portlet** above the top-most `objectSpacer` within the `panelCustomizable`.
3. Select the Rich Text portlet and open the Property Inspector.
4. Set the following properties:

Properties	Value
AllModesSharedScreen	true
IsMinimizable	false
IsMaximizable	false
DisplayHeader	false

5. From the Component Palette, choose **RichTextPortletProducer**, then select **Rich Text** from the list. In the Structure pane, drag the **Rich Text Portlet** below the last `objectSpacer` within the `panelCustomizable`.
6. Select the Rich Text portlet and open the Property Inspector.
7. Set the following properties:

Properties	Value
AllModesSharedScreen	true
IsMinimizable	false
IsMaximizable	false
DisplayHeader	false

8. Save the page.

You can now customize this Rich Text Portlet by going back to the login page after you have logged in. Chapter 9, "Building a Site Administration Page" describes how you can do this.

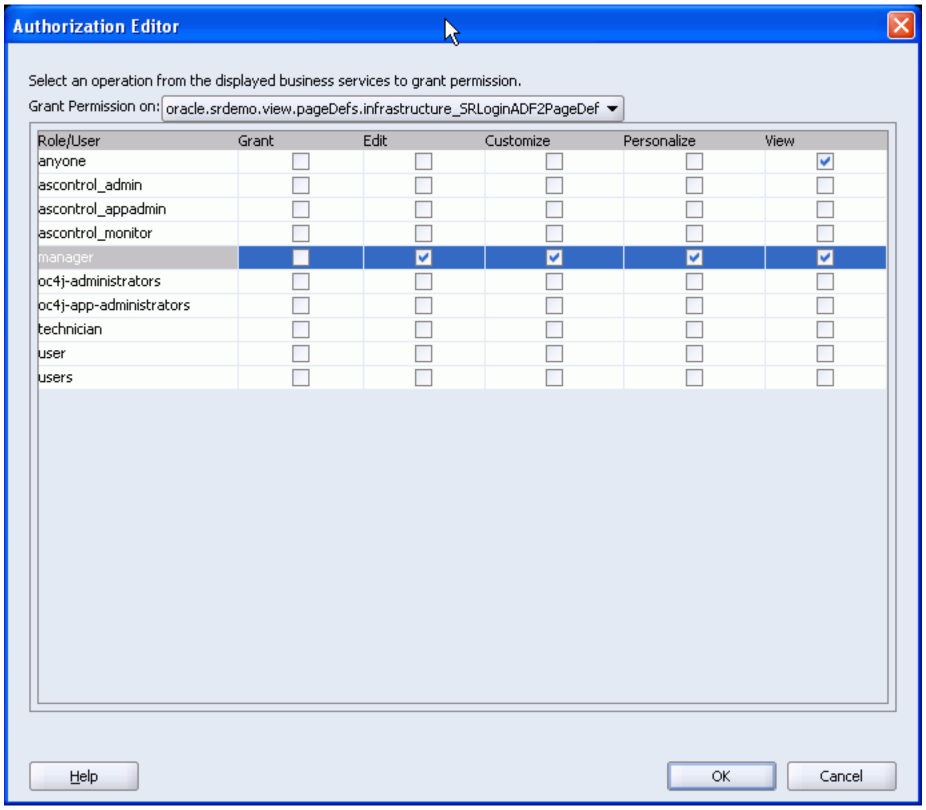
Step 4: Editing Authorization for the Login Page

Next, we need to define the appropriate privileges for the login page. All users should be able to view the page and managers should also be able to edit the page, so that they can update the Rich Text portlet, for example. To set the security on the login page, perform the following steps:

1. In the Applications Navigator, right-click **SRLoginADF.jspx**.
2. Select **Go to Page Definition**.
3. Click **Yes**, if you are prompted to create a new page definition. The page definition file opens in the Structure pane.
4. Right-click the Page Definition file and select **Edit Authorization**. This displays the Authorization Editor.

5. Grant **View** privilege to *anyone* and **Edit** privilege to *manager*, as shown in Figure 6–6.

Figure 6–6 Authorization Editor to define Security for the Login Page



Select an operation from the displayed business services to grant permission.

Grant Permission on: oracle.srdemo.view.pageDefs.infrastructure_SRLoginADF2PageDef

Role/User	Grant	Edit	Customize	Personalize	View
anyone	☐	☐	☐	☐	✓
ascontrol_admin	☐	☐	☐	☐	☐
ascontrol_appadmin	☐	☐	☐	☐	☐
ascontrol_monitor	☐	☐	☐	☐	☐
manager	☐	✓	✓	✓	✓
oc4j-administrators	☐	☐	☐	☐	☐
oc4j-app-administrators	☐	☐	☐	☐	☐
technician	☐	☐	☐	☐	☐
user	☐	☐	☐	☐	☐
users	☐	☐	☐	☐	☐

Step 5: Configuring the Application to Use ADF Security and the Login Page

In this step, you will use the Oracle ADF Security Wizard to configure authentication settings for the SRDemo application. All the options that you select are recorded in `web.xml`. To configure authentication settings, you must perform the following configuration tasks:

- Enable Oracle ADF authentication
- Choose the lightweight XML resource provider for user authentication
- Specify Form-Based as the protocol for authentication
- Grant authenticated users (`ValidUser`) access to the `adfAuthentication` servlet

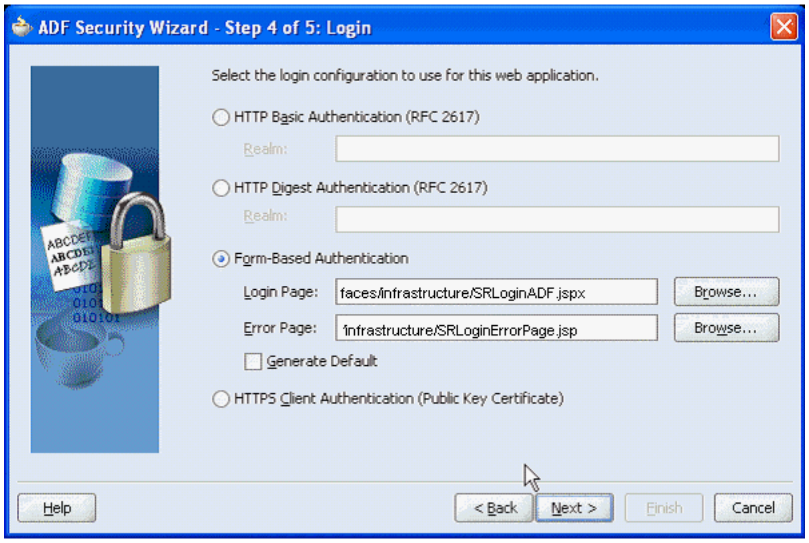
To configure the SRDemo application to use ADF Security and the login page, perform the following steps:

1. In the Applications Navigator, select **UserInterface**.
2. From the **Tools** menu, choose **ADF Security Wizard**. The ADF Security wizard will guide you through the configuration process.
3. If needed, click **Next** to skip the Welcome page.

4. Ensure that **Enforce Authorization** is selected. This option configures the `adfAuthentication` servlet and configures authorization rules (appropriate filters to allow for checking of the current user's permissions on the page).
5. Click **Next** to move to the next page of the wizard.
6. Choose **Lightweight XML Provider**. Oracle ADF Security authenticates users against a given resource provider. For the SRDemo we will use the lightweight resource provider `system-jazn-data.xml` that you copied while following the steps in the `install.html` file located in the sample files you downloaded.
7. Click **Next** to display the next page of the wizard.
8. On this page, set the Location to **Application Repository**, Default Realm to **jazn.com**, JAAS Mode to **doAsPrivileged**, and then click **Next**.
9. On the login page, choose **Form-Based Authentication** (Figure 7-8). This specifies that the SRDemo 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 we will be using the login form we built earlier in `SRLoginADF.jspx`.
10. In the Login Page field, enter `/infrastructure/SRLoginADF.jspx`.
11. In the Error Page field, you can also enter `faces/infrastructure/SRLoginErrorPage.jsp`, as shown in Figure 6–7.

Figure 6–7 Form-Based Authentication Details


ADF Security Wizard - Step 4 of 5: Login

Select the login configuration to use for this web application.

( ) HTTP Basic Authentication (RFC 2617)
Realm:
( ) HTTP Digest Authentication (RFC 2617)
Realm:
(•) Form-Based Authentication
Login Page: faces/infrastructure/SRLoginADF.jspx
Error Page: /infrastructure/SRLoginErrorPage.jsp
☐ Generate Default
( ) HTTPS Client Authentication (Public Key Certificate)

12. Click **Next** to display the final page of the wizard.

    This page defines resources within your application that should be secured, and specifies which J2EE security roles can access each resource. The `adfAuthentication` resource (the authentication servlet) is defined for you. You cannot edit or delete this resource, but you can specify the set of roles that may access this resource. One J2EE role, named `oc4j-administrators`, is selected by default, but we want any valid user to be able to access the `adfAuthentication` resource, so you will need to create another J2EE role, which you will name `ValidUser`, and grant access to this role.
13. Remove the Web resource **J2EE-Secured-Application**, if it shows up.

14. Click **Manage Roles**.
15. Click **Add**, and enter the name `ValidUser`.

    Later on, you will map this J2EE role to one of the identity store roles, `users`, which you defined at the start of this lesson. This 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 a need for the definition of specific permissions.
16. Click **OK**. The `ValidUser` role should appear in the list.
17. Click **Close**.
18. Click the double arrow (**Add All**) to move everything in the Available Roles list to the Selected Roles list.
19. This completes ADF Security wizard settings, so click **Next** and then **Finish**.
20. You can now run the application's `SRWelcome.jspx` page and this should take you to the publicly accessible welcome page.
21. Click the **login** link. This will take you to the login page you built in this chapter.
22. Log in as a manager (`sking`). The contents of the RTP would normally be defined in a site administration page. This page will be built later.

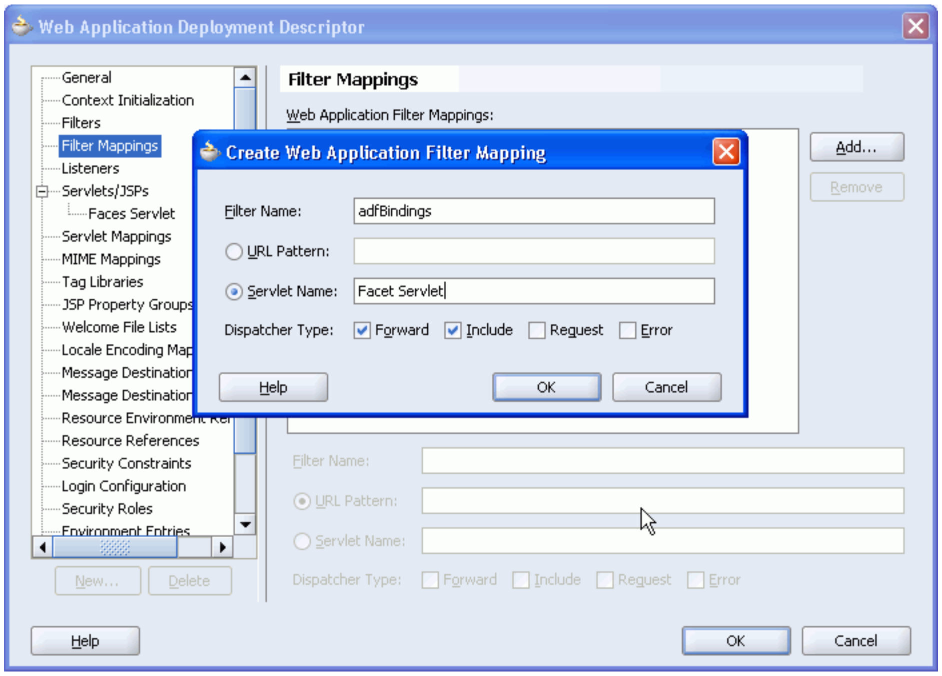
Note: Prior to creating the site administration, you can customize the login page by directly accessing the page URL as a user with Customize privilege on the page.

Step 6: Updating Web.xml to Call the New Login Page

The steps to update `web.xml` to call the new login page are as follows:

1. In the Applications Navigator, expand the **WEB-INF** node and open the `web.xml` property palette (choose Properties from the right-click menu).
2. Select **Filter Mappings** in the left panel and add a new filter mapping for the `adfBindings` Filter as shown Figure 6–8.

Figure 6–8 New Filter Mappings for the adfBindings Filter



Create Web Application Filter Mapping

Filter Name: adfBindings

( ) URL Pattern:

(●) Servlet Name: Facet Servlet

Dispatcher Type: [✓ Forward] [✓ Include] ☐ Request ☐ Error

Help | OK | Cancel

Filter Name:

(●) URL Pattern:

( ) Servlet Name:

Dispatcher Type: ☐ Forward ☐ Include ☐ Request ☐ Error

Make sure that the Servlet Name = `Faces Servlet` and that the Forward and Include dispatcher types are set.

3. Click **OK** to save the changes to `web.xml`.

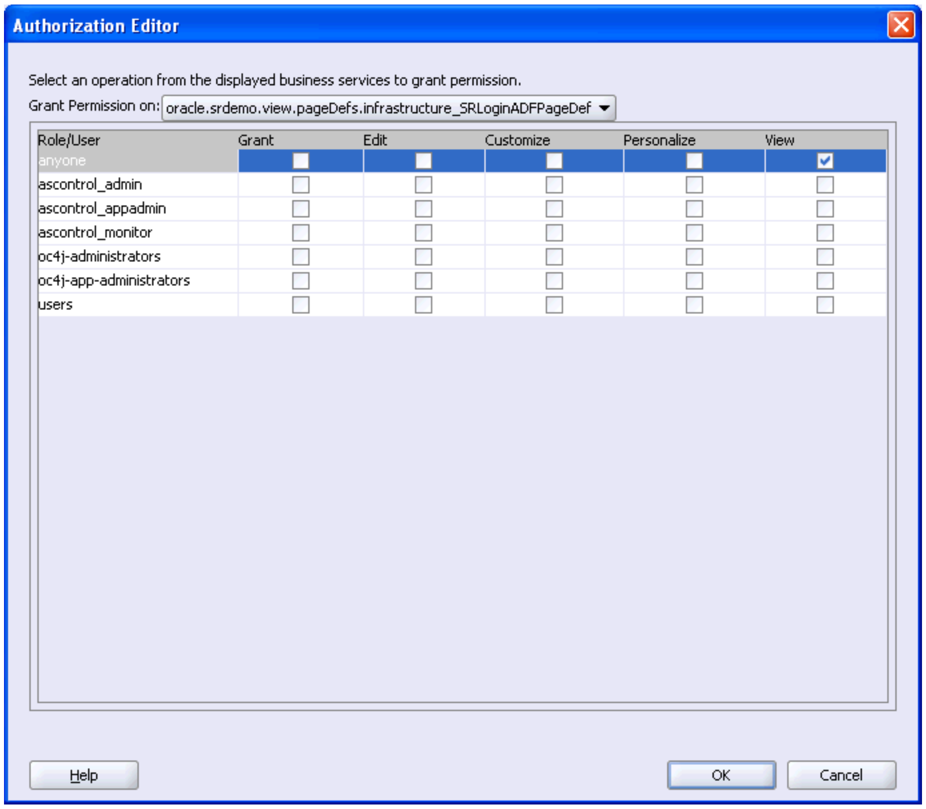
> **Note:** As the login page is also a JSPX page (and hence secured by ADF security) it must be defined as a public page. If the page is not defined as public then the container will continually redirect to the defined authentication point before allowing access to this page (which of course is this page resulting in a continuous loop).

4. Right-click the `SRLoginADF.jspx` page in the application navigator and select **Go to Page Definition**. Confirm creation of the page definition file if it currently does not exist.

5. Right-click the page definition file and select **Edit Authorization**.

6. Grant **View** privilege to the *anyone* role, as shown in Figure 6–9.

Figure 6–9 Authorization Editor



Select an operation from the displayed business services to grant permission.

Grant Permission on: oracle.srdemo.view.pageDefs.infrastructure_SRLoginADFPageDef

Role/User	Grant	Edit	Customize	Personalize	View
anyone	☐	☐	☐	☐	✓
ascontrol_admin	☐	☐	☐	☐	☐
ascontrol_appadmin	☐	☐	☐	☐	☐
ascontrol_monitor	☐	☐	☐	☐	☐
oc4j-administrators	☐	☐	☐	☐	☐
oc4j-app-administrators	☐	☐	☐	☐	☐
users	☐	☐	☐	☐	☐

7. As the login page contains customizable portlets, the appropriate role must have customize privilege on the page.

Step 7: Running the Application

You can now run the application and test whether the login and error pages you created are displayed correctly. To do this, perform the following steps:

1. Run the `SRWelcome.jspx` page.
2. Click the **login** link. This displays the login page.
3. Log in using a valid username and password.

   You are logged in to the demo application successfully.
4. Log out from the application.
5. To test whether the error page is displayed correctly, login using incorrect credentials three times. The error page is displayed.

Summary

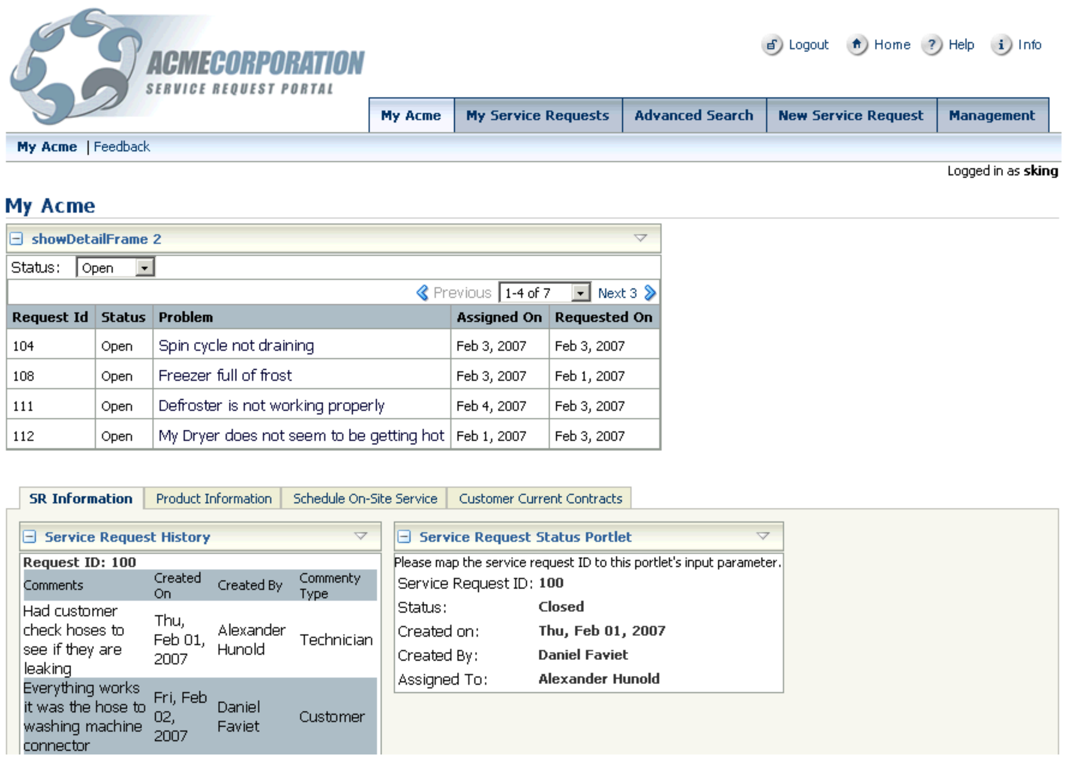
In this chapter, you learned how to build an ADF Faces-based login page and configure the SRDemo application to use this login page. You can now enable administrators to customize the login page at run time. Chapter 9, "Building a Site Administration Page" describes the steps involved in enabling this type of customization.

7

Building a Page and Adding Components

This chapter describes how to build a new page for your application that contains portlets, ADF Faces components, and interactivity between these items on the page, as shown in Figure 7–1.

Figure 7–1 My Acme Page Containing Portlets and ADF Components Wired Together



Logged in as **sking**

My Acme

showDetailFrame 2

Status: Open

Previous 1-4 of 7 Next 3

Request Id	Status	Problem	Assigned On	Requested On
104	Open	Spin cycle not draining	Feb 3, 2007	Feb 3, 2007
108	Open	Freezer full of frost	Feb 3, 2007	Feb 1, 2007
111	Open	Defroster is not working properly	Feb 4, 2007	Feb 3, 2007
112	Open	My Dryer does not seem to be getting hot	Feb 1, 2007	Feb 3, 2007

Service Request History

Request ID: 100

Comments	Created On	Created By	Commenty Type
Had customer check hoses to see if they are leaking	Thu, Feb 01, 2007	Alexander Hunold	Technician
Everything works it was the hose to washing machine connector	Fri, Feb 02, 2007	Daniel Faviet	Customer

Service Request Status Portlet

Please map the service request ID to this portlet's input parameter.

Service Request ID:	**100**
Status:	**Closed**
Created on:	**Thu, Feb 01, 2007**
Created By:	**Daniel Faviet**
Assigned To:	**Alexander Hunold**

Note: Before you can create the elements on this page, ensure you have registered the producers as described in Chapter 2, "Building Your Portlets" and set up your content repository, as described in Chapter 4, "Setting Up Your Content Repository".

This chapter contains the following sections:

- Step 1: Creating the MyAcme Page
- Step 2: Registering the OmniPortlet Producer

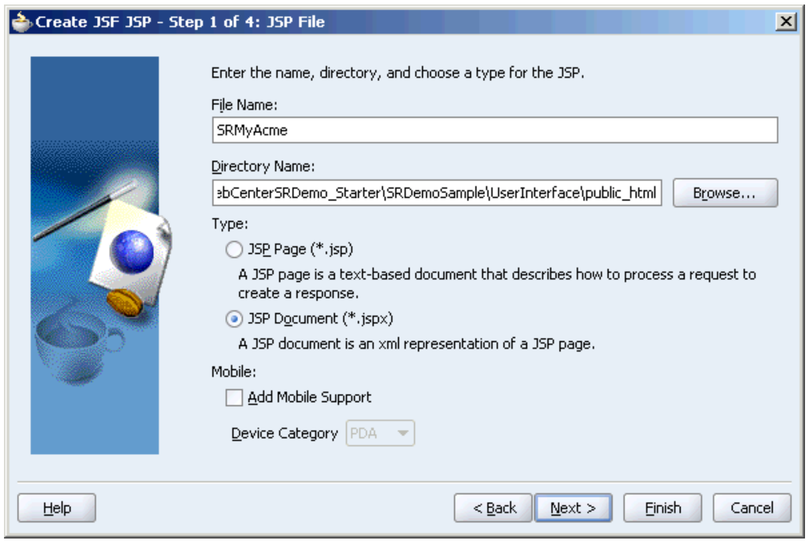
- Step 3: Adding Customization Components
- Step 4: Wiring the SelectOneChoice to the ADF Table
- Step 5: Adding a JavaServer Faces Drop-Down Component
- Step 6: Adding a Service Request History Portlet
- Step 7: Adding the Service Request Status Portlet
- Step 8: Wiring the Table to the Portlets
- Step 9: Adding a Product Details Portlet
- Step 10: Displaying Folder Content
- Step 11: Adding the Schedule On-Site Services Portlet
- Step 12: Adding a Current Contract Based on the User
- Step 13: Applying Security to the My Acme Page
- Step 14: Applying Security to the Components

Step 1: Creating the MyAcme Page

Let's begin by building a page where we will add the components. In this section, we will create the page and apply the template we have provided in the sample files.

1. In the Applications Navigator, under the UserInterface project, expand the **Web Content** folder if necessary.
2. Right-click the **app** folder and select **New**.
3. In the New Gallery, expand the **Web Tier** node.
4. Select **JSF**.
5. In the Items list, select **JSF JSP**.
6. Click **OK** to display the Create JSF JSP dialog box.
7. If you are on the Welcome page of the wizard, click **Next** to display the JSP File page.
8. In the File Name field, enter `SRMyAcme`.
9. Select JSP Document (`*.jspx`), as shown in Figure 7–2.

Figure 7–2 Create the JSF JSP


Create JSF JSP - Step 1 of 4: JSP File

Enter the name, directory, and choose a type for the JSP.

File Name: SRMyAcme

Directory Name: ebCenterSRDemo_Starter\SRDemoSample\UserInterface\public_html

Type:

JSP Page (*.jsp)
A JSP page is a text-based document that describes how to process a request to create a response.

[✓ JSP Document (*.jspx)]
A JSP document is an xml representation of a JSP page.

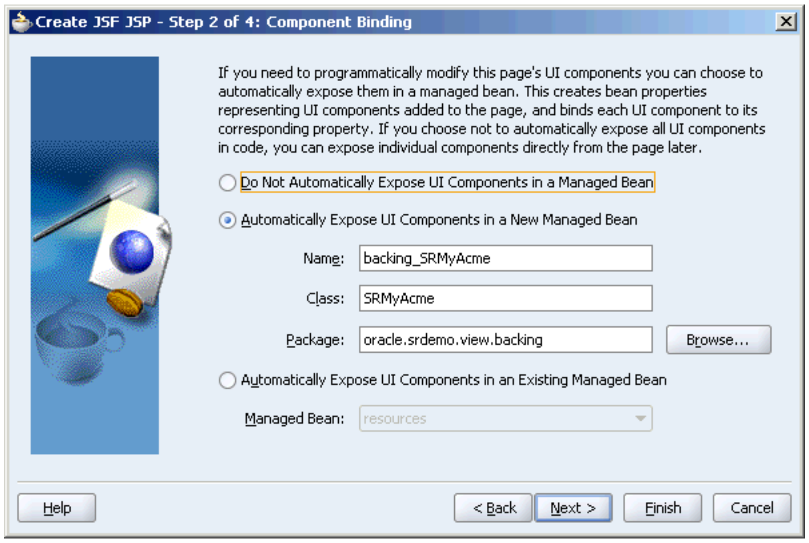
Mobile:

Add Mobile Support

Device Category PDA

10. Click **Next** to display the Component Binding page.
11. On the Component Binding page, select the **Automatically Expose UI Components in a New Managed Bean** option, as shown in Figure 7–3.

Figure 7–3 Create JSF JSP Component Binding


Create JSF JSP - Step 2 of 4: Component Binding

If you need to programmatically modify this page's UI components you can choose to automatically expose them in a managed bean. This creates bean properties representing UI components added to the page, and binds each UI component to its corresponding property. If you choose not to automatically expose all UI components in code, you can expose individual components directly from the page later.

Do Not Automatically Expose UI Components in a Managed Bean

[✓ Automatically Expose UI Components in a New Managed Bean]

Name: backing_SRMyAcme

Class: SRMyAcme

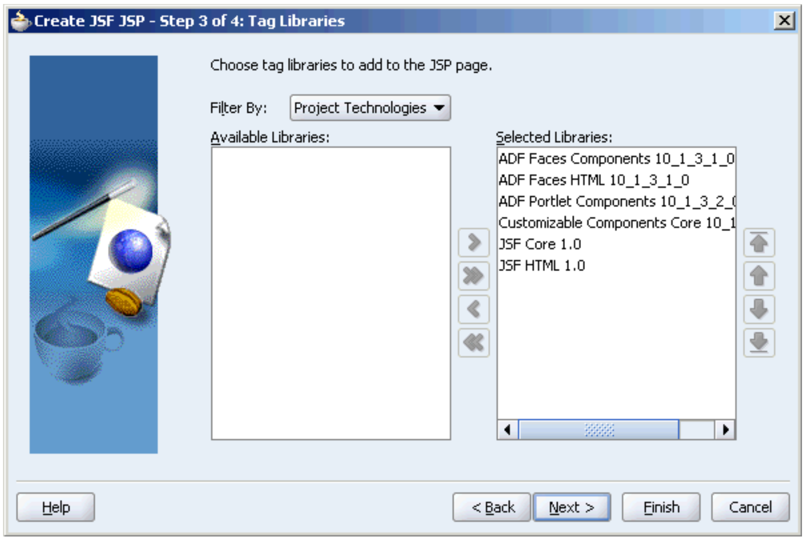
Package: oracle.srdemo.view.backing

Automatically Expose UI Components in an Existing Managed Bean

Managed Bean: resources

12. Click **Next** to display the Tag Libraries page.
13. Filter by Portlet Technologies, then move all the Available Libraries to the Selected Libraries list, as shown in Figure 7–4.

Figure 7–4 Create JSF JSP Tag Libraries List



Choose tag libraries to add to the JSP page.

Filter By: Project Technologies

Available Libraries:

Selected Libraries:
ADF Faces Components 10_1_3_1_0
ADF Faces HTML 10_1_3_1_0
ADF Portlet Components 10_1_3_2_(
Customizable Components Core 10_1
JSF Core 1.0
JSF HTML 1.0

14. Click **Finish** to create the page.
15. Expand the `WEB-INF/template` folder.
16. Right-click **SRDemoTemplate.jspx** and select **Open**.
17. Select and copy the source code from the `SRDemoTemplate.jspx` file and replace the existing code of your new `SRMyAcme.jspx` page.
18. Search for the following line in the source of the `SRMyAcme.jspx` page:

    ```
    <af:panelPage title="#{res['_PAGE_NAME_.pageTitle']}">
    ```

    and replace it with:

    ```
    <af:panelPage title="#{res['SRMyAcmePage.pageTitle']}">
    ```

19. Save the file.
20. Open the `UIResources.properties` file, located under **UserInterface > Application Sources > oracle.srdemo.view > resources**.
21. At the end of the file, add a new entry:

    ```
    #
    ================================================================================
    # Resource Bundle Strings for new WebCenter version of SRDemo
    #
    ================================================================================

    # WC_SRMyAcmePage
    SRMyAcmePage.pageTitle=My Acme
    ```

22. Save the file.

Step 2: Registering the OmniPortlet Producer

In a later step in this chapter, you will create a Service Request History portlet to the My Acme page. You will use OmniPortlet to build this portlet; to do so, you must first register the OmniPortlet producer. You will add the PDK-Java portlet (SRStatus) you created in Chapter 2, "Building Your Portlets" to your page, as well, but if you followed the steps in that chapter, you've already deployed and registered the SRStatus portlet.

Note: For information on registering producers, refer to Chapter 3, "Populating Pages" of the *Oracle WebCenter Framework Developer's Guide*.

To register the OmniPortlet producer:

1. In the Applications Navigator, right-click **UserInterface**, then select **New** from the pop-up menu.
2. In New Gallery, under Categories, expand the **Web Tier** node and select **Portlets**.
3. Under Items, select **Oracle PDK-Java Producer Registration**, then click **OK.**
4. In the Oracle PDK Portlet Producer Registration wizard, on Step 1 of 3, enter the name: `OmniPortlet Producer`, then click **Next**.
5. In the URL Endpoint field, enter the URL of the OmniPortlet Producer, similar to Example 7–1.

Example 7–1 Sample URL Endpoint for the OmniPortlet Producer

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

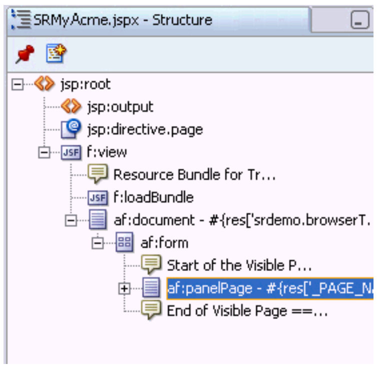
6. Click **Next**.
7. Fill out the remaining options according to your preferences, then click **Finish**.

Step 3: Adding Customization Components

Before you include portlets and content on the `SRMyAcme.jspx` page, you must add certain customizable layout components that allow you to define how these portlets and content are displayed to users.

In the Structure pane of the `SRMyAcme.jspx` page, you will see an `af:panelPage` component as shown in Figure 7–5.

Figure 7–5 PanelPage Component


SRMyAcme.jspx - Structure
jsp:root
jsp:output
jsp:directive.page
f:view
Resource Bundle for Tr...
f:loadBundle
af:document - #{res['srdemo.browserT.
af:form
Start of the Visible P...
af:panelPage - #{res['_PAGE_N
End of Visible Page ==...

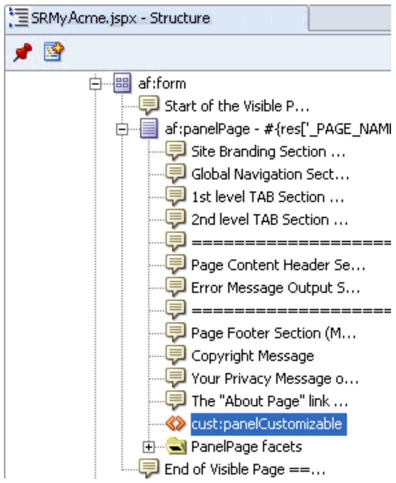
Add the following components within that `PanelPage` component:

1. `PanelCustomizable` Component

Add a `PanelCustomizable` component within the `PanelPage` component to provide the ability to show or hide child components.

To do this, select **PanelCustomizable** from the Customizable Components Core library and drag it on to the `af:panelPage` component in the Structure pane. Set `Layout` to `horizontal`. Figure 7–6 shows the position of the `cust:panelCustomizable` component in the Structure pane.

Figure 7–6 PanelCustomizable Component



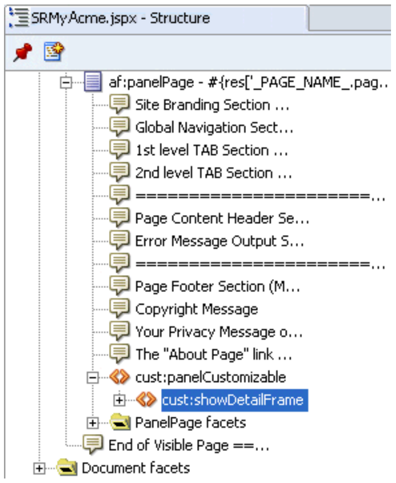
```
SRMyAcme.jspx - Structure
af:form
  Start of the Visible P...
  af:panelPage - #{res['_PAGE_NAMI
    Site Branding Section ...
    Global Navigation Sect...
    1st level TAB Section ...
    2nd level TAB Section ...
    ===================
    Page Content Header Se...
    Error Message Output S...
    ===================
    Page Footer Section (M...
    Copyright Message
    Your Privacy Message o...
    The "About Page" link ...
    cust:panelCustomizable
    PanelPage facets
  End of Visible Page ==...
```

2. `ShowDetailFrame` Component

   Add a `ShowDetailFrame` component within the `PanelCustomizable` component to provide view-level customization of content. For example, options to move, minimize, or maximize the display of content.

   To do this, select **ShowDetailFrame** from the Customizable Components Core library and drag it on to the `cust:panelCustomizable` component in the Structure pane and set `Text` to `My Service Requests`. Figure 7–7 shows the `cust:showDetailFrame` in the Structure pane.

Figure 7–7 ShowDetailFrame Component



```
SRMyAcme.jspx - Structure
  af:panelPage - #{res['_PAGE_NAME_.pag..
    Site Branding Section ...
    Global Navigation Sect...
    1st level TAB Section ...
    2nd level TAB Section ...
    ======================...
    Page Content Header Se...
    Error Message Output S...
    ======================...
    Page Footer Section (M...
    Copyright Message
    Your Privacy Message o...
    The "About Page" link ...
    cust:panelCustomizable
      cust:showDetailFrame
    PanelPage facets
  End of Visible Page ==...
Document facets
```

3. `SelectOneChoice` Component

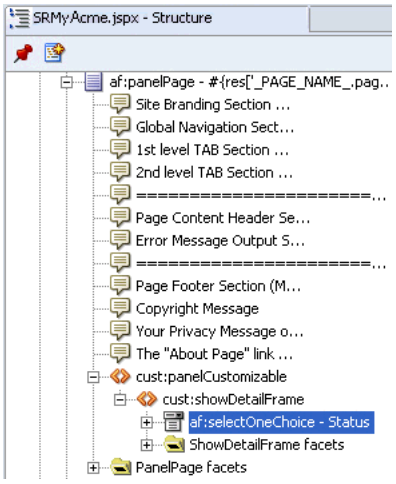
Add a `SelectOneChoice` component within this `ShowDetailFrame` component, to select the status for service requests that must be displayed.

To do this, select **SelectOneChoice** from the ADF Faces Core library and drag it on to the `cust:showDetailFrame` component in the Structure pane. In the Insert SelectOneChoice dialog box displayed, click the appropriate tab and set values for few of the attributes as follows:

- `Label`: `Status`
- `Value`: `Open`
- `AutoSubmit`: `true`
- `Binding`: `#{backing_app_SRMyAcme.statusPicker}`
- `Id`: `statusPicker`

Figure 7–8 shows the `af:selectOneChoice` component in the Structure pane.

Figure 7–8 SelectOneChoice Component


SRMyAcme.jspx - Structure

```
af:panelPage - #{res['_PAGE_NAME_.pag..
  Site Branding Section ...
  Global Navigation Sect...
  1st level TAB Section ...
  2nd level TAB Section ...
  ======================...
  Page Content Header Se...
  Error Message Output S...
  ======================...
  Page Footer Section (M...
  Copyright Message
  Your Privacy Message o...
  The "About Page" link ...
  cust:panelCustomizable
    cust:showDetailFrame
      af:selectOneChoice - Status
      ShowDetailFrame facets
  PanelPage facets
```

4. `SelectItem` Component

Add three `SelectItem` components within the `SelectOneChoice` component, to include components that will be displayed as options under the `SelectOneChoice` component.

To add a `SelectItem` component, select **SelectItem** from the ADF Faces Core library and drag it on to the `af:selectOneChoice` component in the Structure pane.

For each of the new components, set values for few of the attributes as follows:

First `SelectItem` component:

- `Label`: `Open`
- `Value`: `Open`
- `Binding`: `#{backing_app_SRMyAcme.selectItem1}`
- `Id`: `selectItem1`

Second `SelectItem` component:

- `Label`: `Pending`

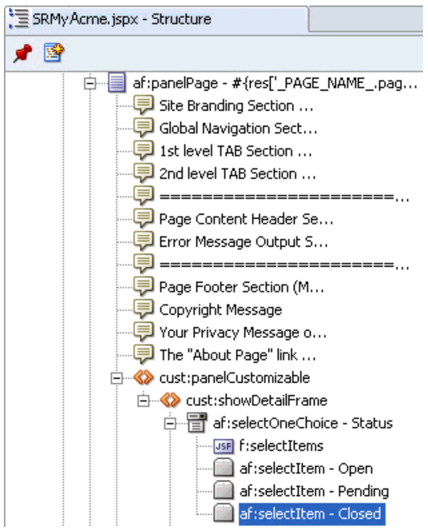
- `Value`: `Pending`
- `Binding`: `#{backing_app_SRMyAcme.selectItem2}`
- `Id`: `selectItem2`

Third `SelectItem` component:

- `Label`: `Closed`
- `Value`: `Closed`
- `Binding`: `#{backing_app_SRMyAcme.selectItem3}`
- `Id`: `selectItem3`

Figure 7–9 shows the `SelectItem` components in the Structure pane.

Figure 7–9 SelectItem Component


SRMyAcme.jspx - Structure

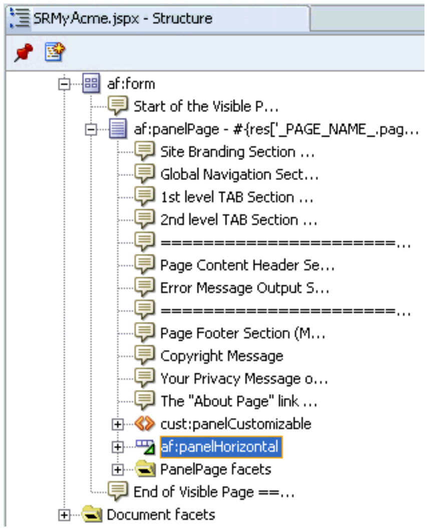
```
af:panelPage - #{res['_PAGE_NAME_.pag...
    Site Branding Section ...
    Global Navigation Sect...
    1st level TAB Section ...
    2nd level TAB Section ...
    ======================...
    Page Content Header Se...
    Error Message Output S...
    ======================...
    Page Footer Section (M...
    Copyright Message
    Your Privacy Message o...
    The "About Page" link ...
    cust:panelCustomizable
        cust:showDetailFrame
            af:selectOneChoice - Status
                f:selectItems
                af:selectItem - Open
                af:selectItem - Pending
                af:selectItem - Closed
```

5. `PanelHorizontal` Component

   Add a `PanelHorizontal` component within the `PanelPage` component to arrange child elements horizontally.

   To do this, select **PanelHorizontal** from the ADF Faces Core component library and drag it on to the `af:panelPage` component in the Structure pane. Ensure that is it located below the `cust:panelCustomizable` component in the Structure pane as shown in Figure 7–10.

Figure 7–10 PanelHorizontal Component



```
SRMyAcme.jspx - Structure
af:form
  Start of the Visible P...
  af:panelPage - #{res['_PAGE_NAME_.pag...
    Site Branding Section ...
    Global Navigation Sect...
    1st level TAB Section ...
    2nd level TAB Section ...
    =====================...
    Page Content Header Se...
    Error Message Output S...
    =====================...
    Page Footer Section (M...
    Copyright Message
    Your Privacy Message o...
    The "About Page" link ...
    cust:panelCustomizable
    af:panelHorizontal
    PanelPage facets
  End of Visible Page ==...
Document facets
```

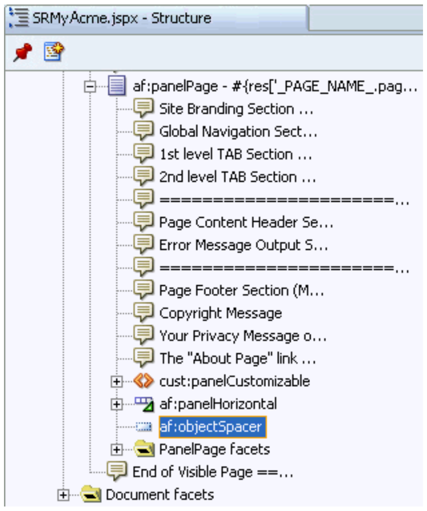
6. `ObjectSpacer` Component

Add an `ObjectSpacer` component (set height to 30 and width to `10`) below the `PanelHorizontal` component to include a fixed amount of space between components in the layout.

To do this, select **ObjectSpacer** from the ADF Faces Core component library and drag it on to the `af:panelPage` component in the Structure pane. Ensure that it is located below the `af:panelHorizontal` component in the Structure pane as shown in Figure 7–11. Set the following attributes:

- `Height`: `30`
- `Width`: `10`

Figure 7–11 ObjectSpacer Component



```
SRMyAcme.jspx - Structure
  af:panelPage - #{res['_PAGE_NAME_.pag...
    Site Branding Section ...
    Global Navigation Sect...
    1st level TAB Section ...
    2nd level TAB Section ...
    =====================...
    Page Content Header Se...
    Error Message Output S...
    =====================...
    Page Footer Section (M...
    Copyright Message
    Your Privacy Message o...
    The "About Page" link ...
    cust:panelCustomizable
    af:panelHorizontal
    af:objectSpacer
    PanelPage facets
  End of Visible Page ==...
Document facets
```

7. `ShowOneTab` Component

Add a `ShowOneTab` within the `PanelPage` component to include a series of items defined by `ShowDetailItem` components.

To do this, select **ShowOneTab** from the ADF Faces Core component library and drag it on to the `af:panelPage` component in the Structure pane. Ensure that it is located below the `af:objectSpacer` component in the Structure pane. Then, select the ShowOneTab and set the `Position` property to `above`.

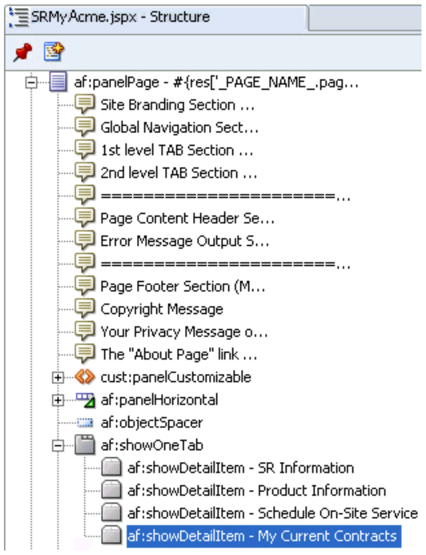
8. `ShowDetailItem` Component

   Add four `ShowDetailItem` components within the `ShowOneTab` to include components that will be displayed as tabs in the `ShowOneTab` component.

   Set the `Text` property for these `ShowDetailItem` components to `SR Information`, `Product Information`, `Schedule On-Site Service`, and `My Current Contracts`. Later, you will add four components within these `ShowDetailItem` components.

   To add a `ShowDetailItem`, select **ShowDetailItem** from the ADF Faces Core component library and drag it on to the `af:showOneTab` component in the Structure pane. Figure 7–12 shows the `af:showOneTab` component with the four child `af:showDetailItem` components in the Structure pane.

Figure 7–12 ShowOneTab Component



SRMyAcme.jspx - Structure

```
af:panelPage - #{res['_PAGE_NAME_.pag...
    Site Branding Section ...
    Global Navigation Sect...
    1st level TAB Section ...
    2nd level TAB Section ...
    ======================...
    Page Content Header Se...
    Error Message Output S...
    ======================...
    Page Footer Section (M...
    Copyright Message
    Your Privacy Message o...
    The "About Page" link ...
    cust:panelCustomizable
    af:panelHorizontal
    af:objectSpacer
    af:showOneTab
        af:showDetailItem - SR Information
        af:showDetailItem - Product Information
        af:showDetailItem - Schedule On-Site Service
        af:showDetailItem - My Current Contracts
```

9. `PanelHorizontal` Component

   Add a `PanelHorizontal` component within the `ShowDetailItem` component titled `SR Information` to arrange child elements horizontally.

   To do this, select **PanelHorizontal** from the ADF Faces Core component library and drag it on to the `cust:showDetailItem component - SR Information` component in the Structure pane.

   Later, you will add an OmniPortlet and an Oracle Java PDK portlet within this component to display the service request history and service request status respectively.

10. Add a `PanelCustomizable` component within the `ShowDetailItem` component titled `Product Information` by performing the steps described earlier in this section. Set `Layout` to `horizontal`. Figure 7–13 shows the new `cust:panelCustomizable` component in the Structure pane.

Figure 7–13 ShowDetailItem Components

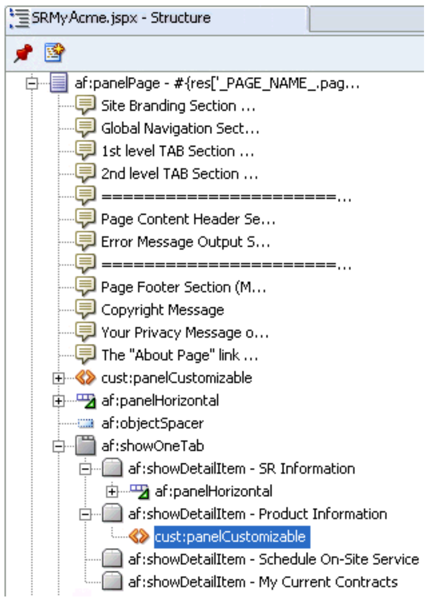
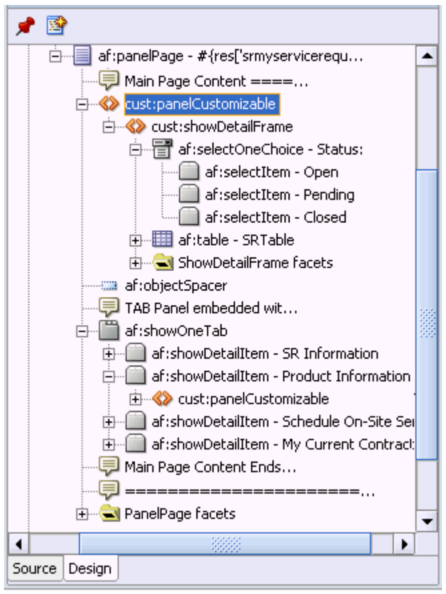

```
SRMyAcme.jspx - Structure
af:panelPage - #{res['_PAGE_NAME_.pag...
  Site Branding Section ...
  Global Navigation Sect...
  1st level TAB Section ...
  2nd level TAB Section ...
  ======================...
  Page Content Header Se...
  Error Message Output S...
  ======================...
  Page Footer Section (M...
  Copyright Message
  Your Privacy Message o...
  The "About Page" link ...
  cust:panelCustomizable
  af:panelHorizontal
  af:objectSpacer
  af:showOneTab
    af:showDetailItem - SR Information
      af:panelHorizontal
    af:showDetailItem - Product Information
      cust:panelCustomizable
    af:showDetailItem - Schedule On-Site Service
    af:showDetailItem - My Current Contracts
```

Figure 7–14 shows how these components appear in the Structure pane of `SRMyAcme.jspx`.

Figure 7–14 Structure Pane Showing Layout Components in the SRMyAcme.jspx Page


```
af:panelPage - #{res['srmyservicerequ...
  Main Page Content ====...
  cust:panelCustomizable
    cust:showDetailFrame
      af:selectOneChoice - Status:
        af:selectItem - Open
        af:selectItem - Pending
        af:selectItem - Closed
      af:table - SRTable
      ShowDetailFrame facets
  af:objectSpacer
  TAB Panel embedded wit...
  af:showOneTab
    af:showDetailItem - SR Information
    af:showDetailItem - Product Information
      cust:panelCustomizable
    af:showDetailItem - Schedule On-Site Se
    af:showDetailItem - My Current Contract
  Main Page Content Ends...
  ======================...
  PanelPage facets
```

The following steps in this chapter provide details about adding portlets and content within these customization components.

Step 4: Wiring the SelectOneChoice to the ADF Table

To display a list of service requests with a particular status, you must add an ADF Read Only Table based on top of Toplink and EJB Session Beans. When a user clicks a service request number in this table, the history and status for that service request are displayed in portlets at the bottom of the page.

To add an ADF table that displays your service requests, perform the following steps:

1. In the Data Control Palette, locate and extract **findServiceRequestByStatus**.

   Tip: If you cannot see the Data Control Palette, select **Data Control Palette** from the View menu.

2. In the Structure Pane, drag **ServiceRequest** on to the `showDetailFrame` component (that you added in the previous section).
3. From the context menu, select **ADF Read Only Table**. This displays the Action Binding Editor. If you see a note about client libraries being added, click **OK**.
4. In the Value field for the `statusParam` parameter, enter `${"Open"}`.

   This hard codes the status of the service request to Open. Later on, we will provide dynamic values.
5. Click **OK**.

   This displays the Edit Table Columns dialog box.
6. Rearrange and delete the columns so that you have the following columns:
   - svrId
   - status
   - problemDescription
   - assignedDate
   - requestDate
7. Select **Enable sorting**.
8. Click **OK**.
9. Right-click **SRMyAcme.jspx** and select **Run**.

   All open service requests are displayed in the table.
10. To reduce the number of rows displayed, go to the page definition and make the change indicated in **bold**:

Example 7–2 Editing the Page Definition for Number of Rows Displayed

```
<methodIterator id="findServiceRequestByStatusIter"
                 Binds="findServiceRequestByStatus.result"
                 DataControl="SRPublicFacade" RangeSize="4"
                 BeanClass="oracle.srdemo.model.entities.ServiceRequest"/>
```

11. Run the page again to see your changes.

Step 5: Adding a JavaServer Faces Drop-Down Component

In the previous section, we had hard coded the status to be `Open`. As a result, you could only view a list of open service requests. Now, we can add a component that allows end users to select the service request status from a list, rather than have this value hard coded. This section shows how to add a list that you can use to select a status and display all service requests with the selected status. To do this, perform the following steps:

1. In the Applications Navigator, expand the **Application Sources** folder.

2. Expand **oracle.srdemo.view > pageDefs** and select **app_ SRMyAcmePageDef.xml**.
3. In the Structure Pane, expand the **bindings** node.
4. Expand the **findServiceRequestByStatus** node.
5. Double-click **statusParam** to display the NamedData Properties dialog box.
6. In the NDValue field, replace the existing value with the following:

   ```
   ${(backing_app_SRMyAcme.statusPicker.value == null) ? "Open" : backing_app_
   SRMyAcme.statusPicker.value}
   ```

7. To ensure the table always reflects the most recent selection of the statusPicker, set the table's `partialTrigger` property to `statusPicker`. To do so, select the table component on the page.

   From the list, select **statusPicker**.
8. In the Property Inspector, locate the Partial Trigger property of the table.
9. Click the **Edit** button to display the Partial Trigger window.
10. Click **New** and select **statusPicker** from the list.
11. Run the page to view your changes.
12. Select a status from the list and see how the table changes.

Step 6: Adding a Service Request History Portlet

The steps in this section show you how to use OmniPortlet to create a portlet that displays the service request history. To create a portlet that displays the service request history using OmniPortlet, perform the following steps:

1. Make sure you have registered the OmniPortlet producer as described in Step 2: Registering the OmniPortlet Producer.
2. In the Component Palette, select **OmniPortlet Producer**, and then select **OmniPortlet**. Drag this on to the `panelHorizontal` component in the Structure pane, located on the SR Information tab.
3. Run the page to view it in a browser.
4. In your browser, click **Define** in your OmniPortlet to change its definition:
   a. On the Type page, choose **SQL**, then click **Next**.
   b. On the Source tab, enter the query shown in Example 7–3.

Example 7–3 SQL Statement for the Service Request History Portlet

```
select svr_id, to_char(svh_date, 'Dy, Mon DD, YYYY') created_on, notes, svh_type,
users.first_name || ' ' || users.last_name createdby
from service_histories, users
where svr_id = substr ('##Param1##', 1, 3)
and users.user_id = service_histories.created_by
order by line_no
```

   c. Edit the Connection, and define the connection to your database.
   d. Specify `104` as the default value of **Param1**, then click **Next** until the View tab is displayed.
   e. On the View tab, make the following changes:

   - **Title**: Service Request History
   - **Header Text**: Request ID: ##Param1##
   - **Layout Style**: Tabular

f. On the Layout tab, set the Tabular Style to **Alternating** and specify the Column Layout as shown in Table 7–1.

Table 7–1 Layout Settings for the Service Request History OmniPortlet

Column Label	Column	Display As
ID	SVR_ID	Hidden
Comments	NOTES	Text
Created by	CREATEDBY	Text
Created on	CREATED_ON	Text
Comment type	SVH_TYPE	Text

5. Click **Finish**.

Note: For more information on OmniPortlet, see Chapter 13, "Creating Portlets with OmniPortlet" in *Oracle WebCenter Framework Developer's Guide*, as well as the steps to building an OmniPortlet in *Oracle WebCenter Framework Tutorial*.

Step 7: Adding the Service Request Status Portlet

In Chapter 2, "Building Your Portlets" you created a portlet that displays the current status of a specific service request, given its service request ID. In this section, you will add that portlet to the MyAcme page.

To add the Service Request Status portlet to a page:

1. Make sure that you have registered the PDK-Java producer as described in Chapter 2, "Building Your Portlets".

   Tip: The producer you registered in Chapter 2, "Building Your Portlets" is the SRStatusProducer, located at `http://host:`*`port`*`/SRStatusPortlet/providers`.

2. In the Component Palette, select **SRStatusProducer** from the list.
3. Drag the **Service Request Status Portlet** onto the `panelHorizontal` component, just below the Service Request History portlet you added earlier.
4. The Service Request Status portlet expects to receive a parameter containing a service request ID. For our page, we want the portlet to display the status of the service request selected in the table at the top of the page. Steps for passing the service request ID to the portlet are provided in Step 8: Wiring the Table to the Portlets.

Step 8: Wiring the Table to the Portlets

Now that you have added all your components and portlets to your page, you can wire them together to create interactivity on the page. In this section, you will set up the page to enable the customer and/or technician to view the status and history of each service request they select from the table on the page. You will also wire the components together using partial page refresh so that the end users will be able to view this information without refreshing the entire page.

In this step, you will do the following:

- Wire the OmniPortlet to the Table
- Wire the Service Request Status and the Service Request History Portlets to the Service Request Table

Wire the OmniPortlet to the Table

While OmniPortlet can display the service request history based on the customization we performed in the previous steps, we now want the portlet to display the history based on a selection that the user makes in the table. The OmniPortlet can receive information from the table and use that information to display the history for a specific service request.

1. In the table you created in Step 4: Wiring the SelectOneChoice to the ADF Table, add a `commandLink` to the problemDescription column of the table.

   In the Component Palette, under ADF Faces Core, choose **commandLink** and drag it onto the Structure pane within the third `af` column and above the `outputText`.

2. Delete the `outputText`.

3. Set the `commandLink` properties as follows:

   - `Text: #{row.problemDescription}`
   - `Id: commandLink1`
   - `Action: #{backing_app_SRMyAcme.commandLink1_action}`

4. Enter the code in Example 7–4 as the `commandLink1_action()` method. You can find this method in the `backing.app.SRMyServiceRequests.java` class, which is located under **Application Source > oracle.srdemo.view > backing.app.SRMyAcme.java**.

Example 7–4 commandLink_action () method

```
public String commandLink1_action() {
        FacesContext context = FacesContext.getCurrentInstance();
        context.getExternalContext().getSessionMap().put("serviceID",
(Integer)outputText1.getValue());
        return "setCurrentRowWithKey";
    }
```

> **Note:** In case `outputText1` is not recognized (it is underlined by Oracle JDeveloper), make sure that the `svrId outputText` column of the table has an ID, and that it is called outputText1. Then, ensure that the binding is defined as `id="outputText1" binding="#{backing_app_SRMyAcme.outputText1`.

5. Expand the import section and add the following line to the top of the Java class:

```
import javax.faces.context.FacesContext;
```

6. In the Structure pane, navigate to the page definition and go to **executables > variable**s, then click **OmniPortlet1_1_Param1**.
7. In the Property Inspector, set the **DefaultValue** property to

   ```
   ${sessionScope.serviceID}
   ```

8. Create a page parameter for OmniPortlet to receive the service ID. In beginning of the page definition, you will see the following:

   ```
   <parameters/>
   ```

   Replace this tag with the following code to set a page parameter called `serviceID`:

   ```
   <parameters>
   <parameter id="pageParam" value="${sessionScope.serviceID}"/>
   </parameters>
   ```

9. Run the page to test the portlet wiring.
10. Click the hyperlinks in the Problem column of the table to test it. Your portlet should be updated every time you click a link.

Wire the Service Request Status and the Service Request History Portlets to the Service Request Table

The Service Request Status and Service Request History portlets both expect to receive a service request ID as a parameter. They then display the status or history of the specific service request identified by that ID. We will set the service request ID to be that of the service request selected in the table at the top of the page.

To wire the Service Request Portlet to the Service Request table:

1. Navigate to the page definition and in the structure pane expand **executables > variables** and click **OmniPortlet1_1_Param1.**
2. In the Property Inspector, set the **DefaultValue** property to `${sessionScope.serviceID}`.
3. In the Structure pane, locate and select the variable **ServiceRequestStatusPortlet1_1_ServiceReqID**.

   Tip: You can find this by expanding the following nodes: **executables > variables > ServiceRequestStatusPortlet1_1_ServiceReqID**.

4. In the Property Inspector, set the **DefaultValue** property to `${sessionScope.serviceID}`.
5. Save the page, then run it to your browser. You can now test the portlet by clicking a service request in the table. The Service Request History and Service Request Status portlets update accordingly.

Step 9: Adding a Product Details Portlet

The steps in this section show you how to display a JSR 168 portlet on the page. Here, you will add the Product Details portlet you created in Chapter 2, "Building Your Portlets". On the Product Information tab, in the `panelCustomizable` component, add the following:

1. Make sure that you have registered the PDK-Java producer as described in Chapter 2, "Building Your Portlets".

   Tip: The producer you registered in Chapter 2, "Building Your Portlets" is the ProductDetailsPortlet, located at `http://`*`host`*`:`*`port`*`/ProductDetailsPortlet/portlets/wsrp1?WSDL`.

2. In the Component Palette, select **ProductDetailPortlet** from the list.
3. Drag the **Product Details Portlet** onto the `panelHorizontal` component.
4. Save the page, then run it to your browser to test the portlet on the Product Information tab.

Step 10: Displaying Folder Content

The steps in this section show you how to display a folder's contents on a page. On the Product Information tab, in the `panelCustomizable` component, add the following:

1. In the Data Control Palette, expand **SRContentRepository**. You created this data control in Chapter 4, "Setting Up Your Content Repository".

   Tip: If you cannot see the Data Control Palette, select **Data Control** Palette from the View menu.

2. Expand **getItems(String, String)**.
3. Add a `panelBox` to the `panelCustomizable`.
4. Drag the **Return** node into the panelBox.
5. From the menu select **Trees**, then **ADF Tree**.

   This displays the Action Binding Editor.
6. In the Value field for the `path` parameter, enter `/manuals`.
7. Leave the Value field for the `type` parameter empty.
8. Click **OK**.

   This displays the Tree Binding Editor.
9. In the Tree binding Editor, select **Name**, URI, and **Primary Type** from the top right list.
10. In the Branch Rule Accessor list, select **Items**.
11. Click **Add New Rule**.
12. Click **OK**.
13. Click **OK**.
14. In the page source, find the code that looks something like:

```
<af:panelBox>
<af:tree value="#{bindings.getItems1.treeModel}" var="node">
<f:facet name="nodeStamp">
<af:outputText value="#{node}"/>
</f:facet>
</af:tree>
</af:panelBox>
```

15. In the Component Palette, select **ADF Faces Core**.
16. Drag the **Switcher** component inside the `<f:facet>` tag.
17. Set the `facetName` attribute to `#{node.currentRow.dataProvider.primaryType}`.
18. Create one facet named `nt:file` and one named `nt:folder`. To do so, right-click **Switcher**, then choose **Insert Inside af:switcher > JSF Core > Facet**.
19. For `nt:folder`, use the output text that was generated.

Note: If the output text is not generated automatically, you may need to add an output text inside the facet, than set the value attribute.

20. For `nt:file`, create a `goLink` and an `objectImage`.
21. The code for the switcher should look something like Example 7–5.

Example 7–5 Code for the Switcher

```
<af:switcher facetName="#{node.primaryType}">
<f:facet name="nt:file">
<h:panelGroup>
<af:objectImage source="/images/file.gif"/>
<af:goLink text="#{node.name}" destination="#{node.URI}" targetFrame="_blank"/>
</h:panelGroup>
</f:facet>
<f:facet name="nt:folder">
<af:outputText value="#{node.name}"/>
</f:facet>
</af:switcher>
```

22. Run `index.html`.
23. Click the **Product Information** tab.
24. Click one of the links in the table.
25. Expand one of the product categories to see the different models.
26. Expand one of the models to view a list of manuals for that model.
27. Select a manual to view.

Step 11: Adding the Schedule On-Site Services Portlet

The Schedule On-Site Services portlet enables you to schedule an on-site service for your product after you have created a service request in the application. By specifying the service ID for your request, and selecting a convenient date and time or a technician of your choice, you can schedule for a technician to come to the specified location and attend to the problem.

This section covers the following topics:

- Registering a Web Clipping Producer
- Adding the Web Clipping Portlet
- Selecting the Web Page to Display in the Web Clipping Portlet

Registering a Web Clipping Producer

To register the Web Clipping producer:

1. In the Applications Navigator, right-click **SRDemoSample_Starter** under which to create the producer and select **New** from the context menu.
2. In New Gallery, under Categories, expand the **Web Tier** node and select **Portlets**.
3. In New Gallery, under Items, select **Oracle PDK-Java Producer Registration**.
4. Click **OK**.
5. On the Welcome page, click **Next**.

   Optionally, before clicking Next, select **Skip this Page Next Time** to forego display of the Welcome page on subsequent uses of this wizard. The Welcome page may not display if the option to skip was selected on earlier use of the wizard.
6. In the **Name** field, enter `WebClipping Producer` as the name for the Web Clipping producer.

   The name must be unique within this application. Use only letters, numbers, and the underscore character.
7. Click **Next**.
8. In the **URL Endpoint** field, enter the URL of the Web Clipping producer as shown in Example 7–6.

Example 7–6 Sample URL Endpoint for the Web Clipping Producer

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

9. Click **Finish** to complete registration of the PDK-Java portlet producer.

Adding the Web Clipping Portlet

To add the On-Site Web Clipping Service, perform the following steps:

1. If the `SRMyAcme.jspx` file is not already open, in the Applications Navigator, right-click the `SRMyAcme.jspx` file, and select **Open** from the context menu.
2. In the Component Palette, select **WebClipping Producer**.
3. Select **WebClippingPortlet**, and drag it on to the `showDetailItem` component called `Schedule On-Site Service` in the Structure pane.
4. Save your changes.
5. Right-click the `SRMyAcme.jspx` file and select **Run** from the context menu.

   This will start the embedded OC4J server, launch your default browser, and display the Web Clipping Portlet. On the resulting page, you will select a Web page that you want to expose in your WebCenter application. You will then use Web Clipping Studio to select a section of the Web page for inclusion.

Selecting the Web Page to Display in the Web Clipping Portlet

In this section, you will see how to display information from the ACME Technician Assignment System external application in your Web Clipping portlet. This external application is deployed to the OC4J instance available as part of the SRDemo application. Using this external application, you can request for an on-site service for your product.

Prior to performing the steps in this section, you must grant appropriate privileges on the page. See Step 13: Applying Security to the My Acme Page for the steps to grant privileges.

To display the ACME Technician Assignment System application in your Web Clipping portlet, perform the following steps:

1. Click the **Actions** icon on the header of the Web Clipping portlet, and select **Customize**, to select a Web page that can be used by all users.

   This displays the Find a Web clipping page.

2. In the **URL Location** field, enter the following URL:

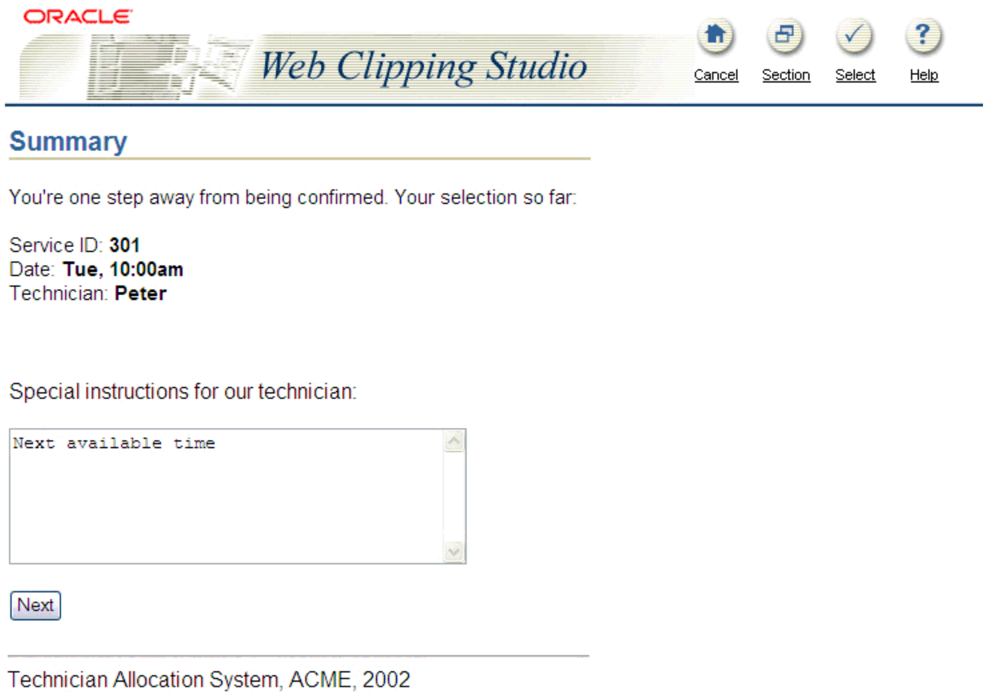
   ```
   http://host:port/ExternalApp/preConfirmation.jsp?date=Tue,%2010:00am&technician
   =Peter
   ```

   Specify the host name and port number of the system where you downloaded the SRDemo application.

3. Click **Start**.

   The Web Clipping Studio displays the page you specified, as shown in Figure 7–15.

Figure 7–15 Web Clipping Portlet with Standalone Application to Be Clipped



Summary

You're one step away from being confirmed. Your selection so far:

Service ID: **301**
Date: **Tue, 10:00am**
Technician: **Peter**

Special instructions for our technician:

Next available time

Technician Allocation System, ACME, 2002

4. Specify `Next Available Time` as a comment and click **Next**.

5. Click **Select** in the Web Clipping Studio banner. The Web Clipping Studio displays the Find a Web clipping page, with the properties of the clipping.

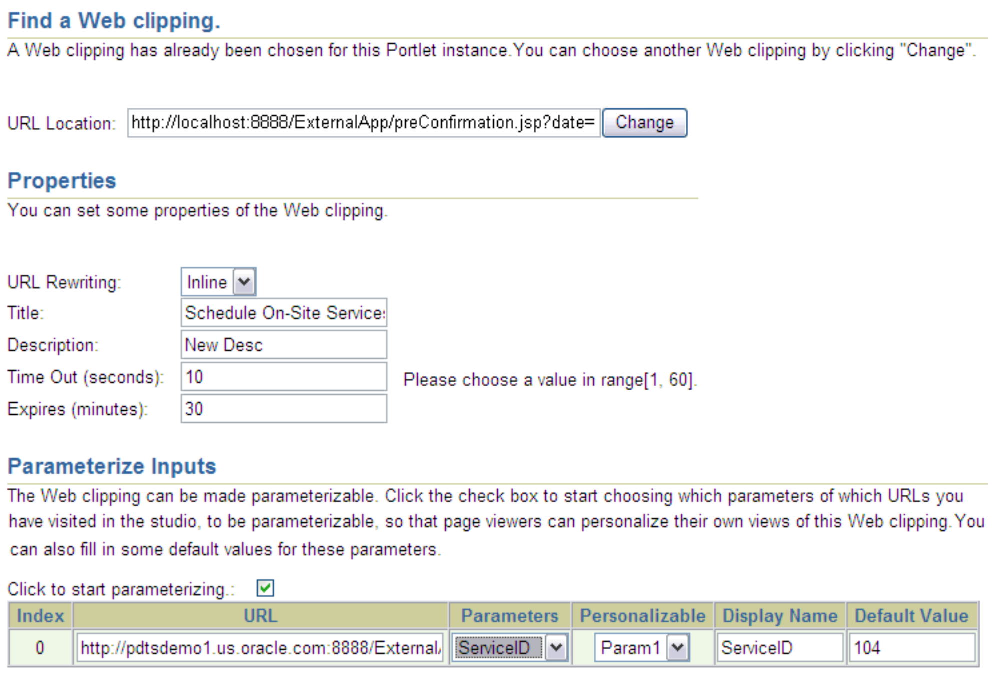
6. In the Find a Web Clipping page shown in Figure 7–16, specify the following:

   - `Title`: `Schedule On-Site Services`

- `Description`: `Embed external scheduling application with the SR application`
- `URL Rewriting`: `Inline`

Note: If you have integrated with an external application or are logged into the clipped site, and if you choose `Inline` for `URL Rewriting`, then the session to the clipped site is maintained while browsing.

Figure 7–16 The Find a Web Clipping Page



Find a Web clipping.

A Web clipping has already been chosen for this Portlet instance. You can choose another Web clipping by clicking "Change".

URL Location: http://localhost:8888/ExternalApp/preConfirmation.jsp?date= Change

Properties

You can set some properties of the Web clipping.

URL Rewriting:	Inline	
Title:	Schedule On-Site Service:	
Description:	New Desc	
Time Out (seconds):	10	Please choose a value in range[1, 60].
Expires (minutes):	30	

Parameterize Inputs

The Web clipping can be made parameterizable. Click the check box to start choosing which parameters of which URLs you have visited in the studio, to be parameterizable, so that page viewers can personalize their own views of this Web clipping. You can also fill in some default values for these parameters.

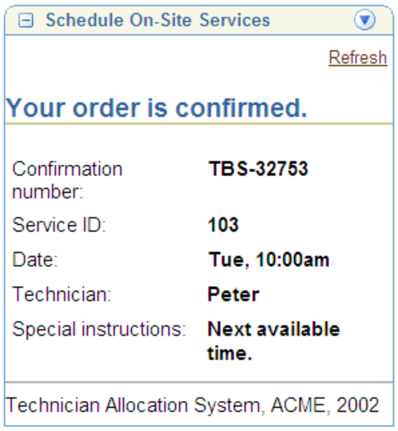
Click to start parameterizing.: ✓

Index	URL	Parameters	Personalizable	Display Name	Default Value
0	http://pdtsdemo1.us.oracle.com:8888/External/	ServiceID	Param1	ServiceID	104

7. For the Parameterize Inputs section, specify the following values:
   - `Parameter`: `ServiceID`
   - `Personalize`: `Param1`
   - `Display`: `ServiceID`
   - `Default Value`: `104`
8. Click **OK** to display the selected Web clipping in the Web Clipping portlet on your page.

   Figure 7–17 shows the content added to the Web Clipping portlet.

Figure 7–17 Web Clipping Portlet with Clipped Content



Your order is confirmed.

Confirmation number:	TBS-32753
Service ID:	103
Date:	Tue, 10:00am
Technician:	Peter
Special instructions:	Next available time.

Technician Allocation System, ACME, 2002

Using this portlet, users can schedule on-site services for products.

Step 12: Adding a Current Contract Based on the User

The steps in this section show you how to add a current contract component to your page.

1. Add a panelBox to the My Current Contracts tab.
2. In the Data Control Palette, expand **SRContentRepository**.

   Tip: If you cannot see the Data Control Palette, select Data Control Palette from the View menu.

3. Expand **getItems(String, String).**
4. Drag the **Return** node into the `panelBox`.
5. From the menu, select **Tables**, then **ADF Read-only Table**.

   This displays the Edit Table Columns dialog box.
6. Delete all the columns except for **name**.
7. Click **OK**.
8. In the page definition file (`app_SRMyAcmePageDef.xml`), under `executables`, duplicate the **methodIterator** tab with the ID `"getItemster"` and modify the code as shown in Example 7–7.

Example 7–7 First Modified Code in the app_SRMyAcmePageDef.xml File

```
<methodIterator id="getContractsIter" Binds="getContracts.result"
DataControl="SRContentRepository"
RangeSize="10" BeanClass="SRContentRepository.getItems_return"/>
```

9. Under `Bindings`, duplicate `<methodAction>` with the ID "`getItems`" and modify the code as shown in Example 7–8

Example 7–8 Second Modified Code in the app_SRMyAcmePageDef.xml File

```
<methodAction id="getContracts" InstanceName="SRContentRepository"
 DataControl="SRContentRepository" MethodName="getItems"
RequiresUpdateModel="true"
```

```
 Action="999" ReturnName="SRContentRepository.methodResults.SRContentRepository_
 getContracts_result">
  <NamedData NDName="path" NDValue="/contracts/${userInfo.userName}"
 NDType="java.lang.String"/>
    <NamedData NDName="type" NDType="java.lang.String"/>
</methodAction>
```

10. Look for `<table id="getItems2"...` and modify it to `<table id="getItems2" IterBinding="getContractsIter">`
11. View the source of the page.
12. Locate the table, then locate the **inputText** element.
13. In the Component Palette, select **ADF Core**.
14. Drag a **GoLink** component next to the outputText for name in the table cell on the page.
15. Copy the expression `#{row.name}`from outputText into the text attribute of **GoLink**.
16. Add destination attribute to GoLink with the value `#{row.URI}.`
17. Add a targetFrame attribute with the value `_blank`.
18. Delete the **outputText** element.
19. Set the access privileges:
    a. Open the page definition.
    b. Under `bindings` in the Structure Pane right-click **getItems2**.
    c. From the context menu, select **Edit authorization**.
    d. In the dialog box, check the **Anyone** box for `SRContentRepository.getItems.name` and all other values in the list.
20. Run **index.html**.

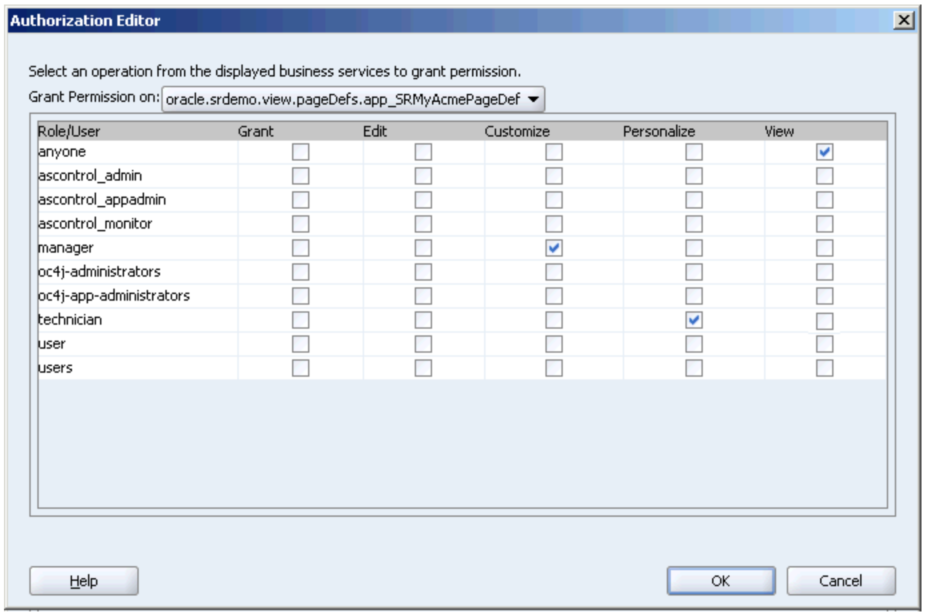
Step 13: Applying Security to the My Acme Page

The page's authorization policies are defined against the Page Definition file. The Authorization Editor is used to read enterprise roles from the local System JAZN file. The editor lets you define the actions that can be set against a given resource, for example, page, method, iterator, and attribute. It writes the permission to the Policy Store and allows for declarative definition of granular JAAS permissions for different roles. Also, the Authorization Editor is used to define a Public page for which the permission is granted to the pseudo role anybody.

Perform the following steps to set up permissions for the `SRMyAcme.jpsx` page:

1. Right-click the **SRMyAcme.jpsx** page and select **Go to Page Definition**.
2. In the Structure window, right-click the page definition and select **Edit Authorization**. This displays the Authorization Editor.
3. Grant View privilege to `anyone`, Customize to `manager`, and Personalize to `technician`, as shown in Figure 7–18.

Figure 7–18 Authorization Editor to Set Privileges on the MyAcme Page



Select an operation from the displayed business services to grant permission.

Grant Permission on: oracle.srdemo.view.pageDefs.app_SRMyAcmePageDef

Role/User	Grant	Edit	Customize	Personalize	View
anyone	☐	☐	☐	☐	✓
ascontrol_admin	☐	☐	☐	☐	☐
ascontrol_appadmin	☐	☐	☐	☐	☐
ascontrol_monitor	☐	☐	☐	☐	☐
manager	☐	☐	✓	☐	☐
oc4j-administrators	☐	☐	☐	☐	☐
oc4j-app-administrators	☐	☐	☐	☐	☐
technician	☐	☐	☐	✓	☐
user	☐	☐	☐	☐	☐
users	☐	☐	☐	☐	☐

Step 14: Applying Security to the Components

On the `SRMyAcme.jspx` page, we want only technicians and managers to be able to view the On-Site Service tab and the Product Details portlet. This section describes how to set these components to be rendered only when a user with the correct permissions is accessing the page.

To apply security to the On-Site Service tab and Product Details portlet, perform the following steps:

1. Open the `SRMyAcme.jspx` page.
2. Select the product details portlet.
3. In the Property Inspector set the `Rendered` property to the following:

```
#{authNLink.authenticated && bindings.permissionInfo['app_
SRMyAcmePageDef'].allowsPersonalize}
```

4. Now select the On-Site Service tab on the `SRMyAcme.jspx` page.
5. In the Property Inspector for this tab set the `Rendered` property to the following:

```
#{authNLink.authenticated && bindings.permissionInfo['app_
SRMyAcmePageDef'].allowsPersonalize}
```

Summary

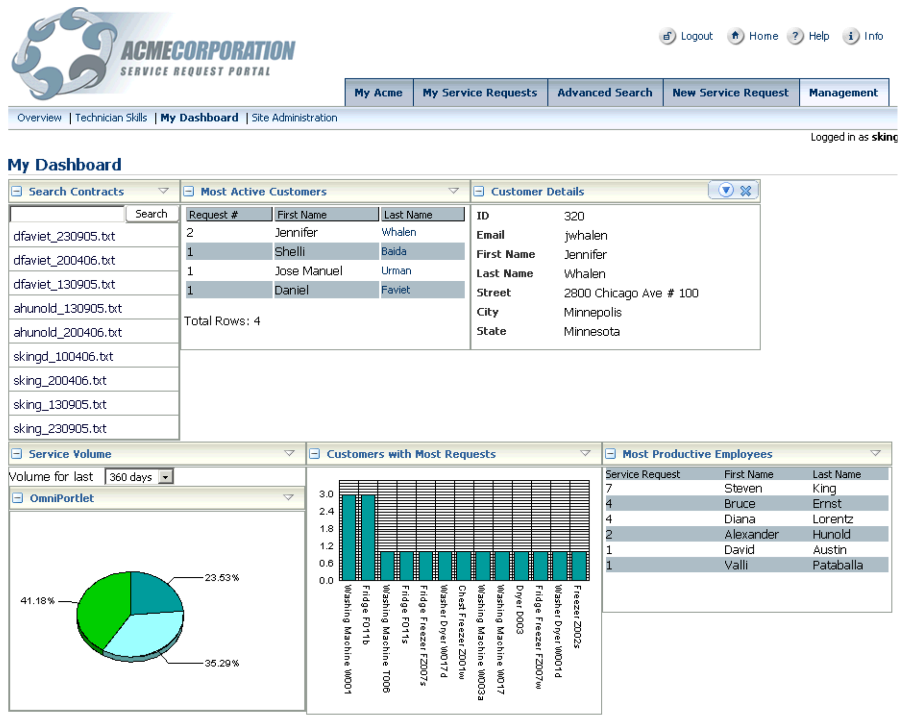
In this chapter, you learned how to add various types of components to a page and organize them into tabs and sub-tabs. You also learned how to apply security to the page and to each of the components.

8 Building a Dashboard Page

In this chapter, you will build a dashboard page for your WebCenter application. A dashboard page is an easy-to-read user interface that organizes and presents metrics and key performance indicators related to business activity and business intelligence.

In this sample dashboard, your end user will be able to view site statistic information and search customer contacts and view volume distribution for each day. The end user can also view information about customers that are the most active, customer details, most productive customers, and products with the most requests, as shown in Figure 8–1.

Figure 8–1 Overview of the Dashboard Page



dfaviet_230905.txt
dfaviet_200406.txt
dfaviet_130905.txt
ahunold_130905.txt
ahunold_200406.txt
skingd_100406.txt
sking_200406.txt
sking_130905.txt
sking_230905.txt

Request #	First Name	Last Name
2	Jennifer	Whalen
1	Shelli	Baida
1	Jose Manuel	Urman
1	Daniel	Faviet

Total Rows: 4

ID	320
Email	jwhalen
First Name	Jennifer
Last Name	Whalen
Street	2800 Chicago Ave # 100
City	Minnepolis
State	Minnesota

Service Request	First Name	Last Name
7	Steven	King
4	Bruce	Ernst
4	Diana	Lorentz
2	Alexander	Hunold
1	David	Austin
1	Valli	Pataballa

This chapter shows you how to build a page and use OmniPortlet to create the dashboard page shown in Figure 8–1 and includes the following sections:

- Step 1: Creating the Dashboard Page
- Step 2: Creating the Page Layout
- Step 3: Adding Instances of OmniPortlet to Your Page

- Step 4: Adding a SelectOneChoice Component for the Service Request Volume Portlet
- Step 5: Adding a Search for Customer Contracts
- Step 6: Defining the Most Productive Employees Portlet
- Step 7: Defining the Service Request Volume Portlet
- Step 8: Defining the Most Requested Products Portlet
- Step 9: Defining the Most Active Customers Portlet
- Step 10: Defining the Customer Details Portlet
- Step 11: Wiring the Page Content Together

Step 1: Creating the Dashboard Page

This section shows you how to build the page on which you will place your portlets to create the dashboard page. Before you begin, ensure that you have the required sample files and initialized your preconfigured OC4J, as described in the `install.html` file located in the sample files.

1. Create a new JSF JSP page, ensuring that you select the following options in the Create JSF JSP Wizard:

   Note: To access the Create JSF JSP Wizard, from the **File** menu, choose **New**, and in the New Gallery dialog box, under **Web Tier**, choose **JSF**.

   - Step 1 of 4: Name the page `SRDashboard.jspx`, point the directory path to `UserInterface\public_html\app\management\`, and select the **JSP Document (*.jspx)** type.
   - Step 2 of 4: Select the **Automatically Expose UI Components in a New Managed Bean** option and leave the remaining defaults on this page.

   Note: Make sure that the name of the New Managed bean is `app_management_SRDashboard`.

   - Step 3 of 4: Ensure that the following libraries are listed in **Selected Libraries**:
     - JSF Core
     - JSF HTML
     - ADF Faces Components
     - ADF Faces HTML
     - ADF Portlet Components
     - Customizable Components Core

   Note: For information on building and populating a page, refer to the **Populating Pages** chapter in the *Oracle WebCenter Framework Developer's Guide*.

2. Once you have completed the wizard and created your page, expand **SRDemoSample_Starter**, **UserInterface**, and **WEB-INF\template** folders of your application.
3. Right-click **SRDemoTemplate.jspx** and choose **Open**.
4. Click the **Source** tab, then copy the source code from the `SRDemoTemplate.jspx` page and paste it into the source of your new `SRDashboard.jspx` page, replacing all the existing code in your new page.
5. Search for `_PAGE_NAME_` and replace it with `srdashboard` so that the `panelPage` tag looks like:

   ```
   <af:panelPage title="#{res['srdashboard.pageTitle']}">
   ```

6. To provide a translatable page title for the new page, expand **SRDemoSample_Starter**, **UserInterface**, **Application Sources**, **oracle.srdemo.view**, **resources**, and then open the **UIResources.properties** file.
7. In the `UIResources.properties` file, add the following lines to the end of file:

   ```
   #WC_DashboardPage
   srdashboard.pageTitle=Dashboard
   ```

8. In the `WEB-INF` folder of the `UserInterface` project, open `faces-config.xml`.
9. If necessary, click the **Source** tab.
10. At the end of the file, before the closing `</faces-config>` tag, add the managed bean shown in Example 8–1 to the existing code to instantiate the bean that manages the menu item for the Dashboard subtab.

Example 8–1 Managed Bean for the Dashboard Subtab

```
<managed-bean>
    <managed-bean-name>subMenuItem_Manage_Dashboard</managed-bean-name>
    <managed-bean-class>oracle.srdemo.view.menu.MenuItem</managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
    <managed-property>
      <property-name>label</property-name>
      <value>#{resources['srdemo.menu.manage.dashboard']}</value>
    </managed-property>
    <managed-property>
      <property-name>shown</property-name>
      <value>#{userInfo.manager}</value>
    </managed-property>
    <managed-property>
      <property-name>viewId</property-name>
      <value>/app/management/SRDashboard.jspx</value>
    </managed-property>
    <managed-property>
      <property-name>outcome</property-name>
      <value>Dashboard</value>
    </managed-property>
<!-- ADF Authorization -->
    <managed-property>
      <property-name>targetPageDef</property-name>
      <value>app_management_SRDashboardPageDef</value>
    </managed-property>
<!-- End ADF Authorization -->
</managed-bean>
```

11. Save the file.
12. Add the following line to the end of the `UIResources.properties` file:

    ```
    srdemo.menu.manage.dashboard=My Dashboard
    ```

13. Open the `faces-config.xml` file and search for `<managed-bean-name>menuItem_Manage`.
14. Now that you have created the subtab, you must add it as a menu item to the parent menu. You can do so by adding the following to the `list-entries` tag:

Example 8–2 Code for Adding the Dashboard Subtab to the Management Page

```
<value>#{subMenuItem_Manage_Dashboard}</value>
```

15. Link the new dashboard menu item to the JSPX page by adding the code shown in Example 8–3 to the navigation rule.

Example 8–3 Code for Linking the Dashboard Subtab to the Dashboard.jspx page

```
<navigation-rule>
   <from-view-id>/app/management/SRManage.jspx</from-view-id>
   <navigation-case>
      <from-outcome>Dashboard</from-outcome>
      <to-view-id>/app/management/SRDashboard.jspx</to-view-id>
   </navigation-case>
</navigation-rule>
```

16. Save the page.

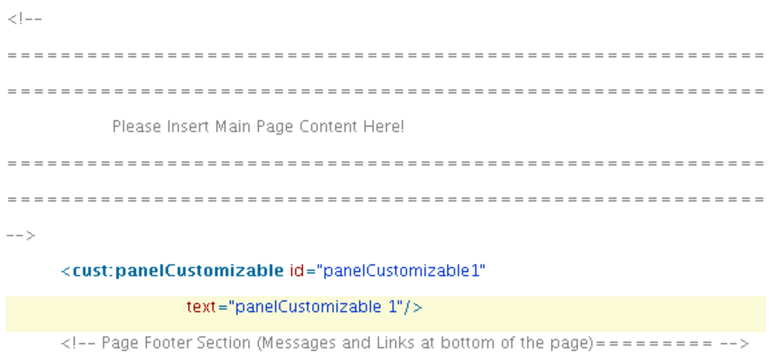
Step 2: Creating the Page Layout

This section shows you how to use customizable components, specifically the `PanelCustomizable` component, to design the layout of your page. For more information on using customizable components, refer to the Customizable Components section in Chapter 7, "Building a Page and Adding Components".

You can view the code of the completed page by opening the `SRDashboard.jspx` page in the completed demo and viewing its source.

1. To add the first row of the dashboard, first open the source of your `SRDashboard.jspx` file.
2. Search for the text "`Please Insert Main Page Content Here!`" and place your cursor below this line.
3. From the Component Palette, choose **Customizable Components Core**.
4. On the Customizable Components Core list, click **PanelCustomizable** to add a `PanelCustomizable` component to your page. The code displays as shown in Figure 8–2.

Figure 8–2 Placement of the First PanelCustomizable

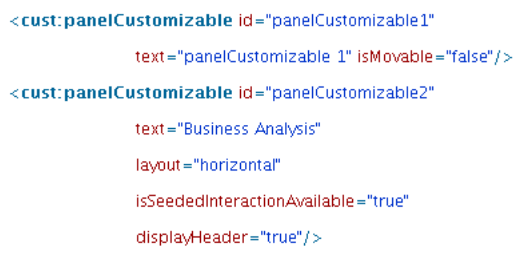


```
<!--
=======================================================
=======================================================
        Please Insert Main Page Content Here!
=======================================================
=======================================================
-->
    <cust:panelCustomizable id="panelCustomizable1"
             text="panelCustomizable 1"/>
    <!-- Page Footer Section (Messages and Links at bottom of the page)========= -->
```

5. In the Structure Pane, select the **cust:panelCustomizable** you just added, and set the `IsMovable` property in the Property Inspector to `false`.
6. Add a second `PanelCustomizable` below the one you just created.
7. In the Structure Pane, select the **cust:panelCustomizable** you just added, and set the following properties in the Property Inspector:
   - `DisplayHeader` to `true`
   - `Id` to `panelCustomizable2`
   - `Layout` to `horizontal`
   - `Text` to `Business Analysis`
   - `IsMovable` to `true`
   - `IsSeededInteraction` to `true`

   The source code of your page should now look like Figure 8–3.

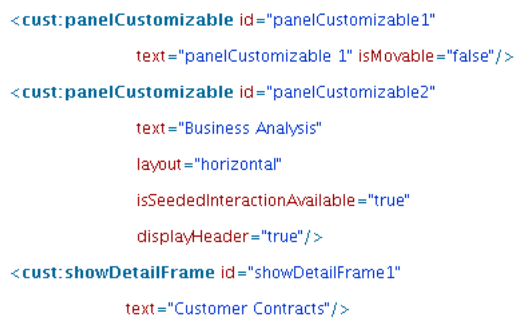
Figure 8–3 PanelCustomizable in the Source Code



```
<cust:panelCustomizable id="panelCustomizable1"
          text="panelCustomizable 1" isMovable="false"/>
<cust:panelCustomizable id="panelCustomizable2"
          text="Business Analysis"
          layout="horizontal"
          isSeededInteractionAvailable="true"
          displayHeader="true"/>
```

8. Create a `ShowDetailFrame` component below the `PanelCustomizable` you just created by choosing **ShowDetailFrame** from the Customizable Components Core Component Palette.
9. Set the `Text` property for the new `ShowDetailFrame` to `Customer Contracts`. The code displays as shown in Figure 8–4.

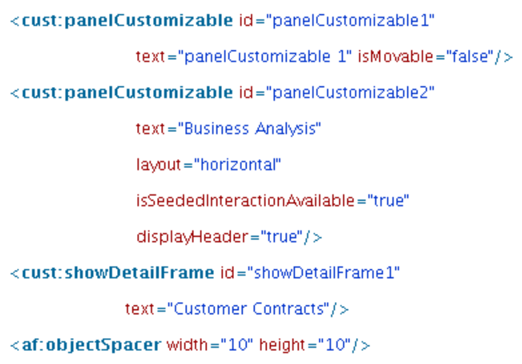
Figure 8–4 ShowDetailFrame in the Source Code



```
<cust:panelCustomizable id="panelCustomizable1"
                text="panelCustomizable 1" isMovable="false"/>
<cust:panelCustomizable id="panelCustomizable2"
                text="Business Analysis"
                layout="horizontal"
                isSeededInteractionAvailable="true"
                displayHeader="true"/>
<cust:showDetailFrame id="showDetailFrame1"
               text="Customer Contracts"/>
```

10. Insert a spacer after the `ShowDetailFrame` component you just created. To do so, choose **ADF Faces Core** from the Component Palette, then select **ObjectSpacer**. The code displays as shown in Figure 8–5.

Figure 8–5 ObjectSpacer in the Source Code



```
<cust:panelCustomizable id="panelCustomizable1"
                text="panelCustomizable 1" isMovable="false"/>
<cust:panelCustomizable id="panelCustomizable2"
                text="Business Analysis"
                layout="horizontal"
                isSeededInteractionAvailable="true"
                displayHeader="true"/>
<cust:showDetailFrame id="showDetailFrame1"
               text="Customer Contracts"/>
<af:objectSpacer width="10" height="10"/>
```

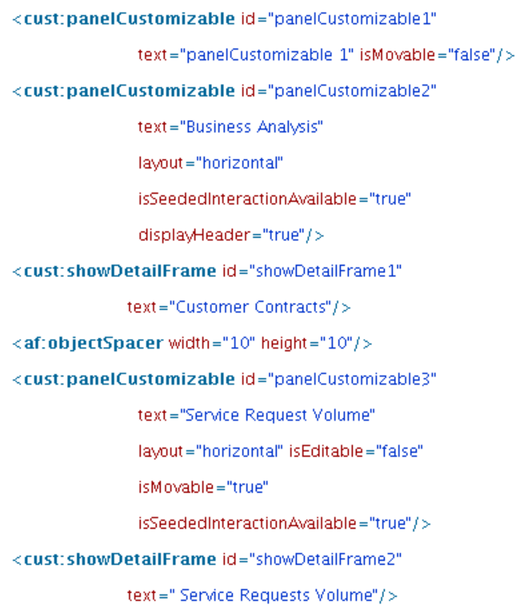
11. Add a second row to your page by creating a second `PanelCustomizable` component with the following properties:

Property	**Value**
DisplayHeader	`false`
Layout	`horizontal`
Text	`Service Requests Volume`
IsMovable	`true`
IsSeededInteractionAvailable	`true`

12. Add a `ShowDetailFrame` with the `Text` property set to `Service Requests Volume`.

    The source code of your page should now look like Figure 8–6.

Figure 8–6 Source Code of Dashboard Page Containing Two Customizable Components



```
<cust:panelCustomizable id="panelCustomizable1"
            text="panelCustomizable 1" isMovable="false"/>
<cust:panelCustomizable id="panelCustomizable2"
            text="Business Analysis"
            layout="horizontal"
            isSeededInteractionAvailable="true"
            displayHeader="true"/>
<cust:showDetailFrame id="showDetailFrame1"
           text="Customer Contracts"/>
<af:objectSpacer width="10" height="10"/>
<cust:panelCustomizable id="panelCustomizable3"
            text="Service Request Volume"
            layout="horizontal" isEditable="false"
            isMovable="true"
            isSeededInteractionAvailable="true"/>
<cust:showDetailFrame id="showDetailFrame2"
           text=" Service Requests Volume"/>
```

13. Next, you will add a horizontal panel inside the `ShowDetailFrame` called "Customer Contracts."

To do so, place your cursor on the line after the `ShowDetailFrame`. From the Component Palette, choose **ADF Faces Core**, then choose **PanelHorizontal** from the list.

14. Next, add an Input Text field inside the `PanelHorizontal`.

To do so, place your cursor on the line after the `PanelHorizontal` you just added, then choose **InputText** from the Component Palette.

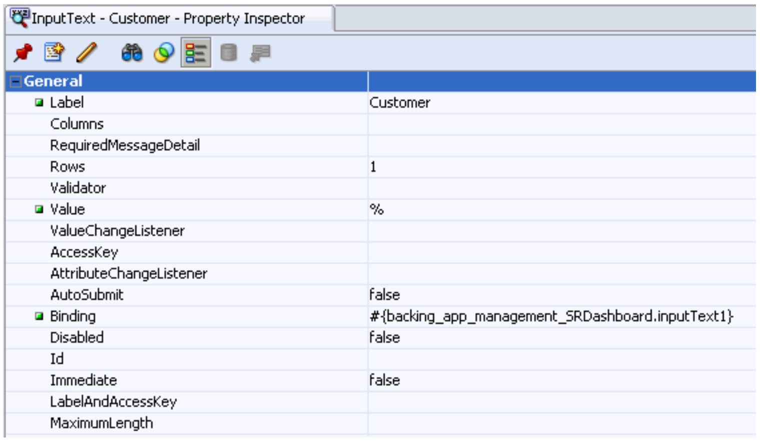
15. Select the `InputText` component in your source.

16. In the Property Inspector, set the **Label** property to `Customer`.

17. Set the **Value** property to: `"%"`.

18. Set the **Binding** property to: `#{backing_app_management_ SRDashboard.inputText1}`. Figure 8–7 shows the Property Inspector with the appropriate values.

Figure 8–7 Property Inspector for the Input Text Field



General	
Label	Customer
Columns	
RequiredMessageDetail	
Rows	1
Validator	
Value	%
ValueChangeListener	
AccessKey	
AttributeChangeListener	
AutoSubmit	false
Binding	#{backing_app_management_SRDashboard.inputText1}
Disabled	false
Id	
Immediate	false
LabelAndAccessKey	
MaximumLength	

The resulting code should look like the source shown in Figure 8–8.

Figure 8–8 Input Text Field Source Code

```
<af:inputText label="Customer" value="%"
        binding="#{backing_app_management_SRDashboard.inputText1}"/>
```

19. Next, add a command button that will enable users to search for customer contracts.

    To do so, place your cursor below the `InputText` component, then choose **CommandButton** from the Component Palette.

20. Set the `Text` property to `Search`.

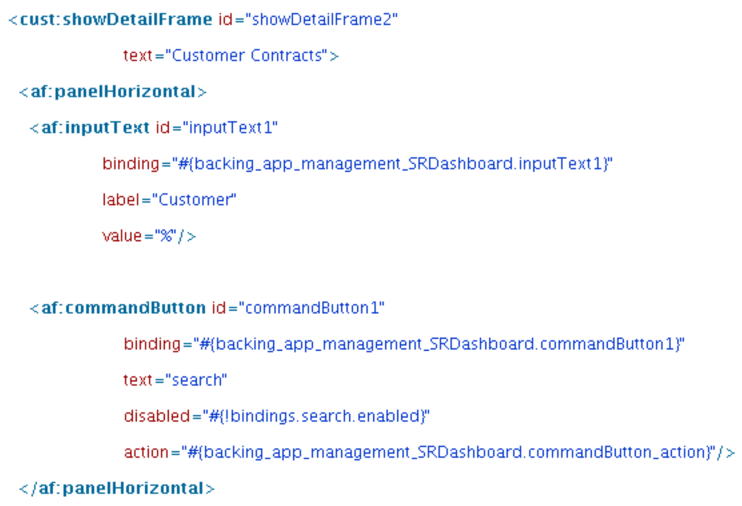
21. Set the `Action` property to: `#{backing_app_management_SRDashboard.commandButton_action}`.

22. In the Property Inspector, set the `Binding` property to: `#{backing_app_management_SRDashboard.commandButton1}`.

23. In the Source, add a `Disabled` property with the definition: `#{!bindings.search.enabled}`.

    Your code should now look like Figure 8–9.

Figure 8–9 Source of the Input Text Field and Command Button



```
<cust:showDetailFrame id="showDetailFrame2"
          text="Customer Contracts">
 <af:panelHorizontal>
  <af:inputText id="inputText1"
         binding="#{backing_app_management_SRDashboard.inputText1}"
         label="Customer"
         value="%"/>

  <af:commandButton id="commandButton1"
          binding="#{backing_app_management_SRDashboard.commandButton1}"
          text="search"
          disabled="#{!bindings.search.enabled}"
          action="#{backing_app_management_SRDashboard.commandButton_action}"/>
 </af:panelHorizontal>
```

24. Add a table below the `InputText` and `CommandButton`, so that the various portlets display horizontally across the page.

Step 3: Adding Instances of OmniPortlet to Your Page

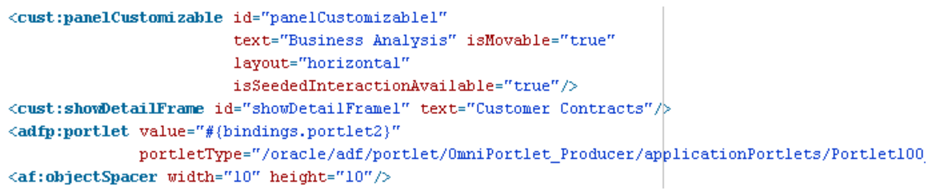
In this section, you will add instances of OmniPortlet to the page layout. You will define the portlets in the later steps of this chapter, and, finally, wire the portlets together.

1. Now that you have created your layout, you can add the portlets to your page.

   In the source of your page, place your cursor after the `ShowDetailFrame` called "Customer Contracts," and before the `ObjectSpacer` you created, then, place your cursor on the first **PanelCustomizable** in the Structure view.

2. From the Component Palette, choose **OmniPortlet Producer**, then click **OmniPortlet**. You should now see the source code of your page, as shown in Figure 8–10.

Figure 8–10 OmniPortlet in the Source Code of Your Page



```
<cust:panelCustomizable id="panelCustomizable1"
                        text="Business Analysis" isMovable="true"
                        layout="horizontal"
                        isSeededInteractionAvailable="true"/>
<cust:showDetailFrame id="showDetailFrame1" text="Customer Contracts"/>
<adfp:portlet value="#{bindings.portlet2}"
             portletType="/oracle/adf/portlet/OmniPortlet_Producer/applicationPortlets/Portlet100
<af:objectSpacer width="10" height="10"/>
```

Note: If you do not see the OmniPortlet Producer listed in the drop-down list of the Component Palette, you may need to register the producers associated with this example.

---

3. Select the OmniPortlet and, in the Property Inspector and set the ID to `customerContracts`.

4. In the source of your page, place your cursor after the OmniPortlet you just added, then add a second instance of OmniPortlet, and change the ID to `mostProductiveEmployees`.
5. In the Structure pane, below the third `PanelCustomizable`, select the `ShowDetailFrame` and add an OmniPortlet, then set the ID to `serviceRequestVolume`.
6. In the Structure pane, select the third `PanelCustomizable` and add another OmniPortlet, then set the ID to `mostRequestedProducts`.
7. Select the third `PanelCustomizable` again and add another OmniPortlet, then set the ID to `customerDetails`.

Step 4: Adding a SelectOneChoice Component for the Service Request Volume Portlet

In this section, you will add a `SelectOneChoice` component for the Service Request Volume portlet.

1. In the Structure view, below the third `PanelCustomizable`, select the **ShowDetailFrame**, then add a `SelectOneChoice` component from the ADF Core Component palette.
2. Set the following properties for the `SelectOneChoice` component:
   - ID = `dayPicker`
   - label = `Volume for last`
   - default value = `360`
   - items = `1,2,3,5,10,30,60,90,180 and 360 days`
   - autoSubmit = `true`

   The following code should appear in the JSPX:

```
<af:selectOneChoice label="Label 1" value="360" autoSubmit="true"
id="dayPicker">
<af:selectItem label="1 day" value="1"/>
<af:selectItem label="2 days" value="2"/>
<af:selectItem label="3 days" value="3"/>
<af:selectItem label="5 days" value="5"/>
<af:selectItem label="10 days" value="10"/>
<af:selectItem label="30 days" value="30"/>
<af:selectItem label="60 days" value="60"/>
<af:selectItem label="90 days" value="90"/>
<af:selectItem label="180 days" value="180"/>
<af:selectItem label="360 days" value="360"/>
</af:selectOneChoice>
```

3. In the Structure view, select the **ShowDetailFrame** from the second `PanelCustomizable`.
4. In the first tab, select **Create list** to fill the content of the drop-down list.
5. Click **Add Item**.
6. In the Item label field, type `1 day` and in the Item Value field, type `1`.
7. Repeat the previous step for the following days: 2, 3, 5, 10, 30, 60, 90, 180, and 360.
8. Click the **Common Properties Tab**.

9. In the Label field, type `Volume for last`.
10. In the Value field, enter `360`.
11. In Advanced Properties tab, for the autoSubmit option, enter `true`.
12. Click **OK** to create the component.
13. In the Property view, for `SelectOneChoice`, update the `id` to `dayPicker`.
14. Within the `ShowDetailFrame`, drag and drop the **dayPicker** component above the OmniPortlet.

Step 5: Adding a Search for Customer Contracts

In the following steps, you will add a component that allows your users to search for existing customer contracts.

1. In the page source, locate the `ShowDetailFrame` called "Customer Contracts" and place your cursor under the `PanelHorizontal` that is located under the `ShowDetailFrame`.
2. In the Data Control palette, expand the **ContentRepository** data control, then expand **Search**.
3. Drag and drop the Search method as an ADF Command Button below the `PanelHorizontal`.
4. In the Structure Pane, select the **Search** method.
5. In the Action Binding Editor, specify "`/contracts`" as the path.
6. Set **isRecursive** to **true**, then click **OK**.
7. In the Data Control palette, under Search, expand **Parameters**.
8. Drag and drop the **NamePattern** method as an Input Text with Label component before the Search Button.
9. In the Structure Pane, select the **InputText** component you just added and set the **Label** property to `Name (use % as wildcard).`
10. In the Data Control palette, under Search, drag and drop the **Return** method as an ADF Read Only table below the `PanelHorizontal`.
11. Save the page.

Step 6: Defining the Most Productive Employees Portlet

In the following steps, you will define your OmniPortlets. Then, at the end of the chapter, you will wire them together to create interactivity.

In this section, you will define the portlet that displays the most productive employees in the organization. To define your OmniPortlet instances, you must first run your JSPX page to your browser, then use the OmniPortlet wizard to define the portlets.

Note: For more information on using the OmniPortlet wizard, refer to the "Creating Portlets with OmniPortlet" chapter in the *Oracle WebCenter Framework Developer's Guide*.

1. Run the `SRDashboard.jspx` page to your browser.

2. Click the **Define** link for the first portlet, and choose **SQL** as the data type, then click **Next**.
3. On the Source tab, enter the SQL statement shown in Example 8–4.

Example 8–4 SQL Statement for the Most Productive Employees Portlet

```
select count(SH.CREATED_BY) as NB_REQ, SH.CREATED_BY,U.FIRST_NAME, U.LAST_NAME
FROM SERVICE_HISTORIES SH INNER JOIN USERS U ON U.USER_ID = SH.CREATED_BY WHERE
SH.SVH_TYPE = 'Technician' OR SH.SVH_TYPE = 'Hidden' GROUP BY SH.CREATED
BY,U.FIRST_NAME,U.LAST_NAME ORDER BY NB_REQ DESC
```

4. Click **Next**, then click **Next** again.
5. On the View tab, select **Tabular**, then click **Next**.
6. On the Layout tab, update the following options:

   Title: `Most Productive Employees`

   Layout: Service Request (NB_REQ)

   First Name (FIRST_NAME)

   Last Name (LAST_NAME)
7. Click **Finish**.

Step 7: Defining the Service Request Volume Portlet

The steps in this section will show you how to display information about the service requests in a pie chart.

1. In your browser, on the `SRDashboard.jspx` page, click the **Define** link for the second portlet.
2. On the Type tab, select **SQL**, then click **Next**.
3. On the Source tab, in the SQL Statement field, enter the following:

Example 8–5 SQL Statement for the Service Request Volume Portlet

```
select count(SVR_ID) as NB_REQ, STATUS FROM SERVICE_REQUESTS
WHERE REQUEST_DATE > (SYSDATE - ##Param1##)
GROUP BY STATUS ORDER BY NB_REQ
```

4. Set up a global connection to the database.

   Under Connection, click **Edit Connection**.
5. In the Connection Name field, type `SRDemo`.
6. In the Username field, type `hr`.
7. In the Password field, type `hr`. These are the username and password for the SRDemo schema you installed when you downloaded the sample files.
8. In the Connection String field, type the path to your database, for example `mydemo.mycompany.com:1522:XE`, then click **OK**. You should now see the connection information in the Connection section of the Source tab.
9. Create a portlet parameter called `Param1` with the default value of `360`, then click **Next**.
10. On the Filter tab, click **Next**.

11. On the View tab, select **Chart**, then click **Next**.
12. On the Layout tab, select **Pie**, and enter `300` for the Width and `200` for the Height.
13. Set the Legend to **Right** and check the **3D effect** checkbox.
14. From the Group list, choose **None**.
15. From the Category list, choose **Status**.
16. From the Value list, choose **NB_REQ**, then click **Finish**.

Step 8: Defining the Most Requested Products Portlet

The steps in this section show you how to define the portlet that displays the most requested products using a SQL statement to access the data in a database.

1. In your browser, on the `SRDashboard.jspx` page, click the **Define** link for the third portlet.
2. On the Type tab, select **SQL**, then click **Next**.
3. On the Source tab, in the SQL statement field, enter the following:

Example 8–6 SQL Statement for the Products Most Requested Portlet

```
select count(SR.SVR_ID) as NB_REQ, SR.PROD_ID, P.NAME FROM
SERVICE_REQUESTS SR INNER JOIN PRODUCTS P ON SR.PROD_ID = P.PROD_ID
GROUP BY SR.PROD_ID, P.NAME ORDER BY NB_REQ DESC
```

4. Click **Next**.
5. On the Filter tab, click **Next**.
6. On the View tab, choose **Chart**, then click **Next**.
7. On the Layout tab, choose **Bar**, then set the following options:

   Width: `300`

   Height: `250`

   Group: `Name`

   Category: `<none>`

   Value: `NB_REQ`
8. Click **Finish**.

Step 9: Defining the Most Active Customers Portlet

The steps in this section will show how to define a portlet that displays the most active customers.

To create the two portlets in the Customer Information section of your page:

1. In your browser, on the `SRDashboard.jspx` page, click the **Define** link for the fourth portlet.
2. On the Type tab, choose **SQL**, then click **Next**.
3. On the Source tab, enter the following SQL statement, then click **Next**.

Example 8–7 SQL Statement for the Most Active Customers Portlet

```
select count(SH.CREATED_BY) as NB_REQ, SH.CREATED_BY , U.FIRST_NAME, U.LAST_NAME
FROM SERVICE_HISTORIES SH INNER JOIN USERS U ON U.USER_ID = SH.CREATED_BY WHERE
SH.SVH_TYPE = 'Customer' GROUP BY SH.CREATED_BY,U.FIRST_NAME,U.LAST_NAME ORDER BY
NB_REQ DESC
```

4. On the Filter tab, make no changes and click **Next**.
5. On the View tab, choose the **HTML Layout** and enter a title: `Most Active Customers`, then click **Next**.
6. On the Layout tab, choose the **Sortable Template**, then click **Apply**.
7. Leave the default code in the Heading Section.
8. In the Repeating Section, update the code so that it looks like the following:

```
<tr class="portlet-section-alternate opRowColorPI1146522302846_##OP_ROWNUM_
MOD2##">
<td class=PortletText1>##NB_REQ##</td>
<td class=PortletText1>##CREATED_BY##</td>
<td class=PortletText1>##FIRST_NAME##</td>
<td class=PortletText1><a href="/SRDemo/faces/app/management/SRDashboard.jspx?
customerID=##CREATED_BY##" target="_top">##LAST_NAME##</a></td>
</tr>
```

9. In the Footer Section, update the code so that it looks like the following:

```
</tbody>
</table>
<font class=PortletText1>Total Rows: ##OP_ROWNUM##</font>
```

10. Click **Finish**.

Step 10: Defining the Customer Details Portlet

In this section, you will define a portlet that displays details about a particular customer.

1. In your browser, on the `SRDashboard.jspx` page, click the **Define** link for the fifth portlet.
2. On the Type tab, choose **SQL**, then click **Next**.
3. On the Source tab, enter the following SQL statement:

Example 8–8 SQL Statement for the Customer Details Portlet

```
select * from USERS where USER_ID = '##Param1##')
```

4. Set **Param1** to the default value `320`, then click **Next**.
5. On the View tab, choose the **HTML Layout** and enter a title: `Customer Details`, then click **Next**.
6. On the Layout tab, clear the fields.
7. Choose **Sortable Table** from the Template drop-down list, and enter the following code in the Heading Section:

```
<TABLE BORDER='0' WIDTH="100%">
```

8. Replace the code in the Repeating Section with the following:

```
<TR CLASS='PortletText1'>
<TD CLASS='PortletHeading1'>ID</TD>
<TD>##USER_ID##</TD>
</TR>
<TR CLASS='PortletText1'>
<TD CLASS='PortletHeading1'>Email</TD>
<TD>##EMAIL##</TD>
</TR>
<TR CLASS='PortletText1'>
<TD CLASS='PortletHeading1'>First Name</TD>
<TD>##FIRST_NAME##</TD>
</TR>
<TR CLASS='PortletText1'>
<TD CLASS='PortletHeading1'>Last Name</TD>
<TD>##LAST_NAME##</TD>
</TR>
<TR CLASS='PortletText1'>
<TD CLASS='PortletHeading1'>Street</TD>
<TD>##STREET_ADDRESS##</TD>
</TR>
<TR CLASS='PortletText1'>
<TD CLASS='PortletHeading1'>City</TD>
<TD>##CITY##</TD>
</TR>
<TR CLASS='PortletText1'>
<TD CLASS='PortletHeading1'>State</TD>
<TD>##STATE_PROVINCE##</TD>
</TR>
```

9. Replace the code in the Footer Section with the following:

```
</tbody>
</table>
<font class=PortletText1>Total Rows: ##OP_ROWNUM##</font>
```

10. Click **Finish**.

Step 11: Wiring the Page Content Together

The steps in this section show you how to add interactivity to the portlets you've just created. In this section, you will create a page parameter and portlet parameters, then wire the portlets together on the page.

1. To create the page parameter, return to Oracle JDeveloper and open the file **SRDashboardPageDef.xml**.
2. Create a page parameter called **custID**.

   To do so, in the Structure Pane, right-click **Parameters**, then insert a new page parameter with the following:

   `<parameter id ="custID" value="$param.customerID)"/>`
3. Open the source for the `SRDashboard.jspx` page you created.
4. Add a portlet parameter to the Customer Details portlet.
5. For `Param1` set the value to the page parameter value from the bindings.custID (set the default value to `320` so that at least a customer shows up and the page is activated and no customers are selected):

```
<parameter name="Param1" value="${(bindings.custID == null || bindings.custID
== '') ? 320 : bindings.custID }"/>
```

6. Add a parameter for the Service Request Volume portlet so that the portlet is refreshed when the value is selected from the `SelectOneChoice`.

   In the Structure view, right-click `parameters`.
7. Select **Insert Inside Parameters**, then select **parameter**.
8. Set the value of **Param1** to `360`. By default we use 1 year or 360 days to show all the results.
9. Select **OmniPortlet3_1_Param1** within executables/variables of the page defs.
10. In the Property Inspector, in the DefaultValue field, type the following:

```
${( app_management_SRDashboard.dayPicker.value == null) ? 360 : app_management_
SRDashboard.dayPicker.value}
```

11. In the Structure pane, choose the fifth portlet, the Most Active Customers portlet.
12. In the Property Inspector, set the `SubmitUrlParameters` property to `true`. Doing so enables the portlet to submit the parameter through a URL to the page.
13. Save your page.
14. Run the `SRDashboard.jspx` page to your browser to view the portlets you just added.

Summary

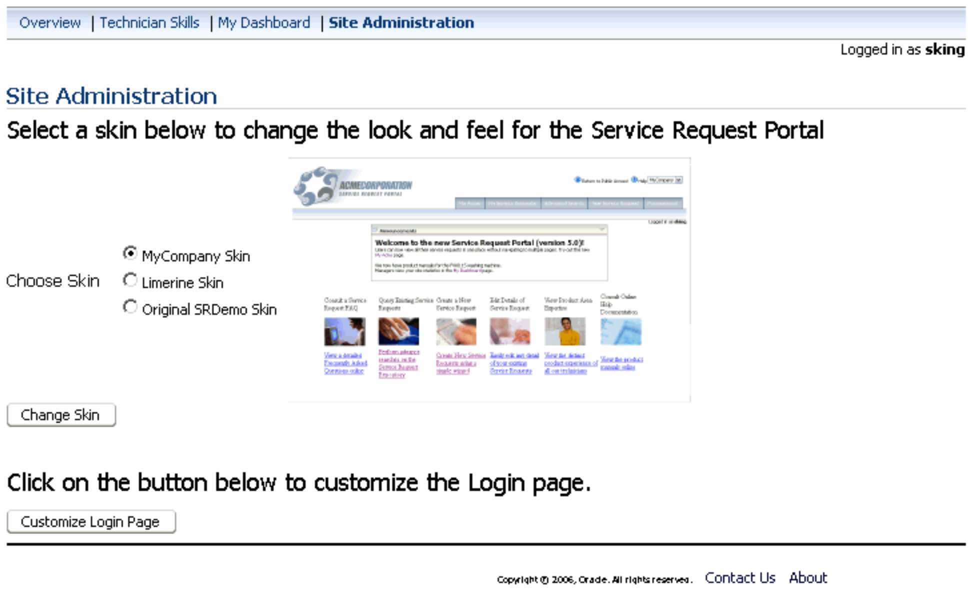
The steps in this chapter showed you how to build a dashboard page for your WebCenter application. You learned how to create a page using customizable components and ADF Faces Core components to create the layout. You also learned how to create portlets that display related information and wire them together on a page, so that your end users can easily view information about their customers for a single entry point.

9 Building a Site Administration Page

You have already created your WebCenter application. In this chapter, you will create a site administration page that can be used by the administrator to change the look and feel of the application and to customize the login page. This site administration page is then added as a subtab of the Management page.

The SRDemo ZIP file contains three skins, `myCompany`, `limerine`, and `original`, and the corresponding image files you may require. Using the site administration page that you create in this chapter, end users will be able to change the skin of the application at run time by choosing from the three skins. In addition, they will be able to customize the login page of the application. At run time, the site administration page would look like Figure 9–1.

Figure 9–1 The Site Administration Page at Run Time


Overview | Technician Skills | My Dashboard | **Site Administration**
Logged in as **sking**
Change Skin
Customize Login Page
Copyright © 2006, Oracle. All rights reserved. Contact Us About

Site Administration

Select a skin below to change the look and feel for the Service Request Portal

Choose Skin [✓ MyCompany Skin] Limerine Skin Original SRDemo Skin

Click on the button below to customize the Login page.

Before you can perform the tasks in this chapter, you must set up skins for your application by performing the steps in Chapter 3, "Setting Up Skins".

This chapter contains the following sections:

- Step 1: Creating the Site Administration Page
- Step 2: Adding the Skin Selector to the Page
- Step 3: Enabling Customization of the Login Page

- Step 4: Adding the Site Administration Page as a Subtab of the Management Page
- Step 5: Using the Site Administration Page for Customization

Step 1: Creating the Site Administration Page

The next step is to build the `SRSiteAdmin.jspx` page with controls that will let administrators change the look and feel of the application and customize the login page.

Note: This page will set the value of the skin to be used in the skin bean we created earlier.

To create the site administration page, perform the following steps:

1. In the Applications Navigator, expand the **UserInterface** node and the **Web Content** node, if necessary.
2. Under the **app** node, right-click the **management** folder and select **New**.
3. In the New Gallery, expand the **Web Tier** node.
4. Select **JSF**.
5. In the Items list, select **JSF JSP**.
6. Click **OK** to display the Create JSF JSP dialog box.
7. If you are on the Welcome page of the wizard, click **Next** to display the JSP File page.
8. In the File Name field, enter `SRSiteAdmin.jspx`.
9. Select **JSP Document (*.jspx)**.
10. Click **Next** to display the Component Binding page.
11. Select **Automatically Expose UI Components in a New Managed Bean** and ensure that the Name field displays `app_management_SRSiteAdmin`.
12. Click **Next** to display the Tag Libraries page.
13. Make sure that the following libraries are listed under Selected Libraries:
    - JSF Core
    - JSF HTML
    - ADF Faces Components
    - ADF Faces HTML
    - ADF Portlet Components
    - Customizable Components Core
14. Click **Finish** to create the page.
15. Expand the **WEB-INF** node and the **template** node.
16. Right-click **SRDemoTemplate.jspx** and select **Open**.
17. Copy the source code from the `SRDemoTemplate.jspx` page and paste it into the source of your new `SRSiteAdmin.jspx` page, replacing all the existing code in your new page.

18. Search for the following line in the source of the `SRSiteAdmin.jspx` page:

```
<af:panelPage title="#{res['_PAGE_NAME_.pageTitle']}">
```

and replace it with:

```
<af:panelPage title="#{res['SRSiteAdmin.pageTitle']}">
```

19. Save the file.

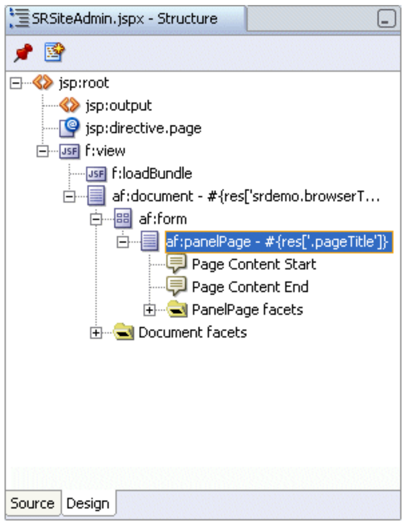
Step 2: Adding the Skin Selector to the Page

Now we must provide the capability to select a specific skin for the application. We will do this by creating a radio group where the administrator can select a skin to use at run time. We will also provide a preview of what each skin looks like.

To add a skin selector to the page, perform the following steps:

1. In the Structure pane, expand the nodes, `f:view`, `af:document`, `af:form`, and `af:panelPage`, as shown in Figure 9–2.

Figure 9–2 Expanded Structure Pane



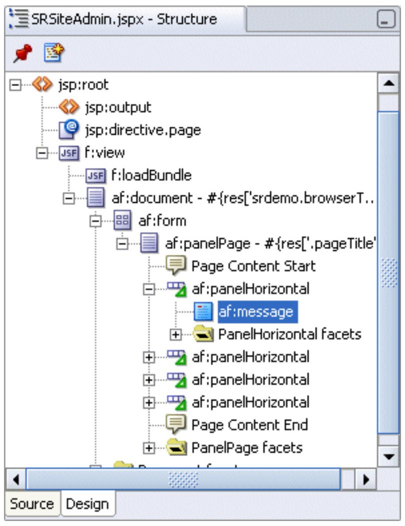
2. In the Component Palette, select **ADF Faces Core**.

3. Drag a **PanelHorizontal** component into the Structure pane so that it appears between `Page Content Start` and `Page Content End`.

4. Add three more `panelHorizontal` components in the same way, making sure that they are all at the same level, and are child components of the `panelPage` component.

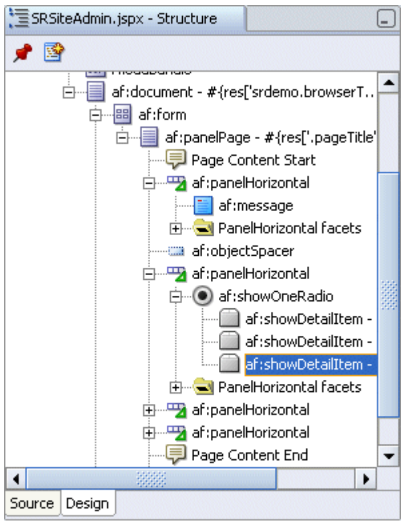
5. Drag a **Message** component from the Component Palette into the Structure pane so that it appears within the first `panelHorizontal` component, as shown in Figure 9–3.

Figure 9–3 Message Component in the Structure Pane of Your Page



6. In the Property Inspector, change the properties for the `Message` component as follows:
   - `Message`: `Select a skin below to change the look and feel for the Service Request Portal`
   - `InlineStyle/font-size`: `large`
7. Drag an **ObjectSpacer** component from the Component Palette into the Structure pane so that it appears between the first and second `panelHorizontal` component.
8. Drag a **ShowOneRadio** component into the second `panelHorizontal` component.
9. Add three `showDetailItem` components under the `showOneRadio` component as shown in Figure 9–4.

Figure 9–4 ShowOneRadio Component In the Structure Pane of Your Page

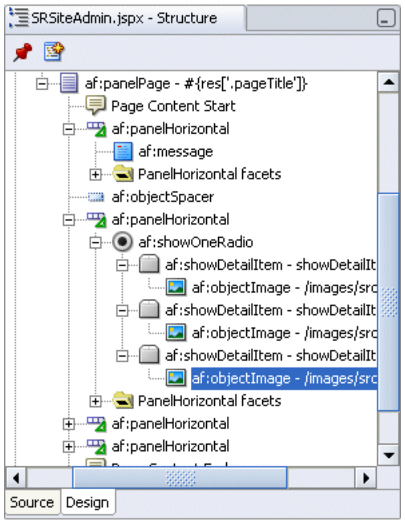


10. Drag an **ObjectImage** component into the first `showDetailItem` component. This displays the Insert ObjectImage dialog box.

11. Next to the Source field, click the **Show Advanced Editor** button.
12. Select **srdemo_mycompany.gif** from the list, and click **OK.**
13. Click **OK.**
14. Repeat the preceding four steps for the other two `showDetailItem` components to add the following images:
    - `srdemo_limerine.gif`
    - `srdemo_original.gif`

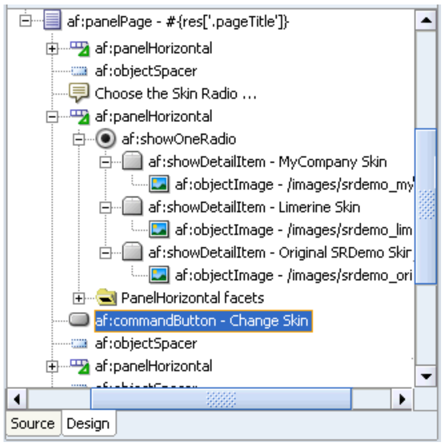
    These components are displayed in the Structure pane as shown in Figure 9–5.

Figure 9–5 Images for the Radio Buttons


SRSiteAdmin.jspx - Structure
af:panelPage - #{res['.pageTitle']}
Page Content Start
af:panelHorizontal
af:message
PanelHorizontal facets
af:objectSpacer
af:panelHorizontal
af:showOneRadio
af:showDetailItem - showDetailIt
af:objectImage - /images/src
af:showDetailItem - showDetailIt
af:objectImage - /images/src
af:showDetailItem - showDetailIt
af:objectImage - /images/src
PanelHorizontal facets
af:panelHorizontal
af:panelHorizontal
Source
Design

15. Select the first `showDetailItem` component and, in the Property Inspector, set the following properties:
    - `Binding: #{backing_SRSiteAdmin.showDetailItem1}`
    - `DisclosureListener: #{skinBean.processDisclosure}`
    - `Text: My Company Skin`
16. Select the second `showDetailItem` component and set the following properties:
    - `Binding: #{backing_SRSiteAdmin.showDetailItem2}`
    - `DisclosureListener: #{skinBean.processDisclosure}`
    - `Text: Limerine Skin`
17. Select the third `showDetailItem` component and set the following properties:
    - `Binding: #{backing_SRSiteAdmin.showDetailItem3}`
    - `DisclosureListener: #{skinBean.processDisclosure}`
    - `Text: Original SRDemo Skin`
18. Drag a **CommandButton** component between the second and third `panelHorizontal` components as shown in Figure 9–6.

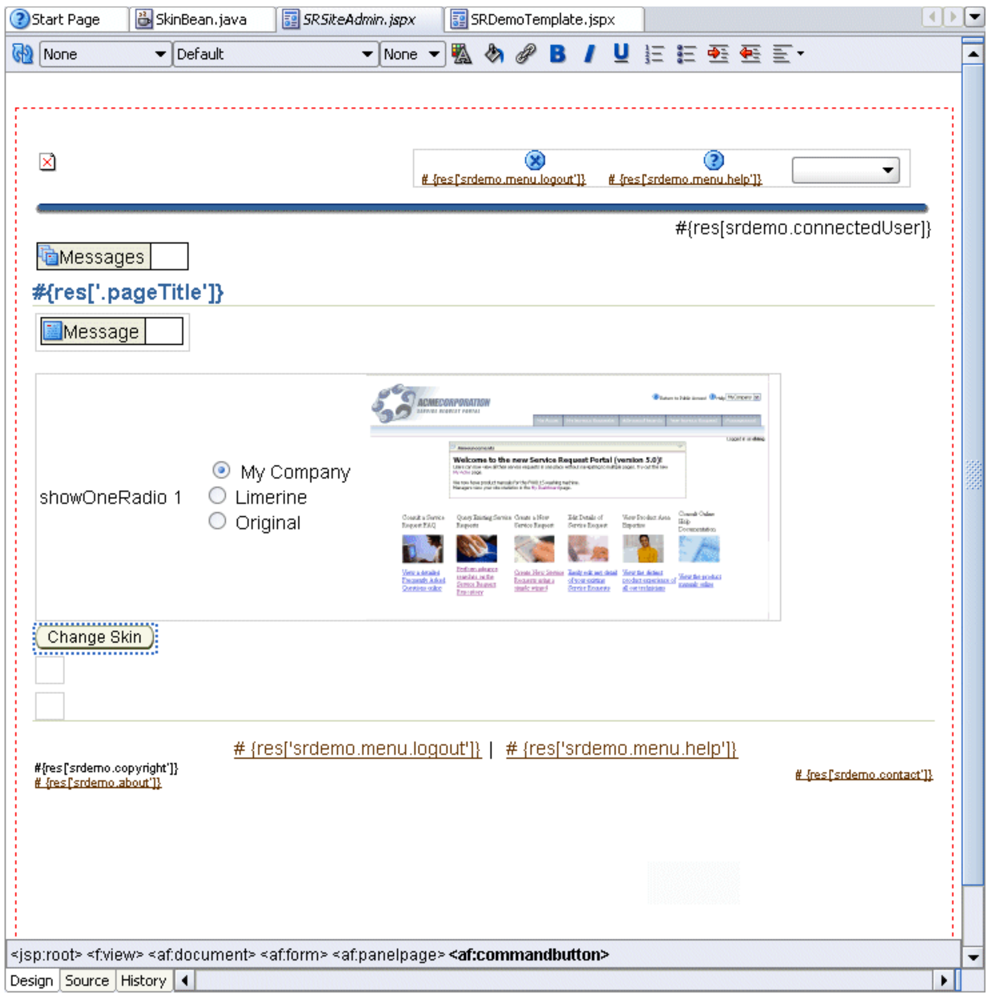
Figure 9–6 Change Skin Command Button



```
af:panelPage - #{res['.pageTitle']}
  af:panelHorizontal
  af:objectSpacer
  Choose the Skin Radio ...
  af:panelHorizontal
    af:showOneRadio
      af:showDetailItem - MyCompany Skin
        af:objectImage - /images/srdemo_my
      af:showDetailItem - Limerine Skin
        af:objectImage - /images/srdemo_lim
      af:showDetailItem - Original SRDemo Skir
        af:objectImage - /images/srdemo_ori
    PanelHorizontal facets
  af:commandButton - Change Skin
  af:objectSpacer
  af:panelHorizontal
```

19. Set the `Text` property of the `commandButton` to `Change Skin`.
20. Drag an **ObjectSpacer** component from the Component Palette into the Structure pane so that it appears between the first and second `panelHorizontal` components.
21. Save the file. The site administration page now looks like Figure 9–7.

Figure 9–7 Site Administration Page


Start Page
SkinBean.java
SRSiteAdmin.jspx
SRDemoTemplate.jspx
None
Default
None
# {res['srdemo.menu.logout']}
# {res['srdemo.menu.help']}
#{res[srdemo.connectedUser]}
Messages
#{res['.pageTitle']}
Message
showOneRadio 1
My Company
Limerine
Original
Change Skin
# {res['srdemo.menu.logout']} | # {res['srdemo.menu.help']}
#{res['srdemo.copyright']}
# {res['srdemo.about']}
# {res['srdemo.contact']}
<jsp:root> <f:view> <af:document> <af:form> <af:panelpage> <af:commandbutton>
Design
Source
History

Step 3: Enabling Customization of the Login Page

You will now see how to customize the login page to include information that can be updated at run time, without redeploying the application. By performing the steps in this section, you can go back to the login page after you have logged in and customize the Rich Text Portlet that you added in "Step 3: Adding a Rich Text Portlet to the Login Page" in Chapter 6, "Creating a Login Page".

To enable customization of the login page, perform the following steps:

1. Create a navigation case to navigate from the site administration page to the login page. To do this, add the code in Example 9–1 to the `faces-config.xml` file:

Example 9–1 Navigation Rule to Navigate from Site Administration Page to Login Page

```
<navigation-rule>
  <from-view-id>/app/management/SRSiteAdmin.jspx</from-view-id>
  <navigation-case>
    <from-outcome>CustomizeLoginPage</from-outcome>
    <to-view-id>/infrastructure/SRLoginADF.jspx</to-view-id>
    <redirect/>
  </navigation-case>
</navigation-rule>
```

2. Create a navigation case to navigate from the login page to the site administration page. To do this, add the code in Example 9–2 to the `faces-config.xml` file:

Example 9–2 Navigation Rule to Navigate from Login Page to Site Administration Page

```
<navigation-rule>
  <from-view-id>/infrastructure/SRLoginADF.jspx</from-view-id>
  <navigation-case>
    <from-outcome>ReturnToSiteAdmin</from-outcome>
    <to-view-id>/app/management/SRSiteAdmin.jspx</to-view-id>
    <redirect/>
  </navigation-case>
</navigation-rule>
```

3. Drag an **ObjectSpacer** component between the `commandButton` and the third `panelHorizontal`.
4. Drag an **OutputFormatted** component into the third `panelHorizontal` and set the following properties in the Property Inspector:
   - `Value`: `#{res['srsiteadmin.editLoginPagePrompt']}`

     This EL returns the appropriate string from the resource bundle.
   - `InlineStyle/font-size`: `large`
5. Drag a **commandButton** into the fourth `panelHorizontal` component and set the following properties:
   - `Text`: `Customize Login Page`
   - `Action`: `CustomizeLoginPage`

   This button will use the previously defined ADF Faces navigation to take the user to the login page.

   > **Note:** Unlike the case where a public user accesses the login page to authenticate, when accessing the page from the site administration page, the user's identity is already defined. Therefore, to prevent a subsequent login attempt while customizing the login page, the login component on the screen must be deactivated. This deactivation is handled by the backing bean code that was added in "Step 1: Creating a Login Page" in Chapter 6, "Creating a Login Page".

6. To prevent all managers from being able to customize the login page, and to expose the button only to those who have customize privilege on the page, the `Rendered` property for the `commandButton` must be set as follows:

   ```
   Render: #{bindings.permissionInfo['SRLoginADFPagePageDef'].allowsCustomize}
   ```
7. Save the file.
8. Open the `SRLogin.jspx` page and drag a **commandButton** on the page and set the following properties:
   - `Text`: `Return`
   - `Action`: `ReturnToSiteAdmin`

   This button will use the previously defined ADF Faces navigation to take the user back to the site administration page after performing the required customizations.

9. Set the `Rendered` property for the `commandButton` as follows:

   ```
   Render: #{authNLink.authenticated}
   ```

10. Save the file.

Step 4: Adding the Site Administration Page as a Subtab of the Management Page

Finally, you must add the `SRSiteAdmin.jspx` page as a subtab of the main Management page. To do this, you must perform the following steps:

1. In the WEB-INF folder of the UserInterface project, right-click **faces-config.xml** and select **Open**.
2. If necessary, click the **Source** tab.
3. Add the managed bean shown in Example 9–3 to the existing code after the managed bean for the Dashboard subtab.

Example 9–3 Managed Bean for the Site Administration Subtab

```
<managed-bean>
    <managed-bean-name>subMenuItem_Manage_SiteAdmin</managed-bean-name>
    <managed-bean-class>oracle.srdemo.view.menu.MenuItem</managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
    <managed-property>
      <property-name>label</property-name>
      <value>#{resources['srdemo.menu.manage.siteadmin']}</value>
    </managed-property>
    <managed-property>
      <property-name>shown</property-name>
      <value>#{userInfo.manager}</value>
    </managed-property>
    <managed-property>
      <property-name>viewId</property-name>
      <value>/app/management/SRSiteAdmin.jspx</value>
    </managed-property>
    <managed-property>
      <property-name>outcome</property-name>
      <value>SRSiteAdmin</value>
    </managed-property>
<!-- ADF Authorization -->
    <managed-property>
      <property-name>targetPageDef</property-name>
      <value>app_management_SRSiteAdminPageDef</value>
    </managed-property>
<!-- End ADF Authorization -->
</managed-bean>
```

4. Open the `UIResources.properties file`, located under **UserInterface**, **Application Sources**, **oracle.srdemo.view**, **resources**.
5. Locate the Admin page area and include the following:

   ```
   srdemo.menu.manage.siteadmin=Administration
   ```

6. Search for `<managed-bean-name>menuItem_Manage.`
7. Add the code in Example 9–4 to the `list-entries` tag.

Example 9–4 Code to Be Added to the list-entries Tag

```
<value>#{subMenuItem_Manage_SiteAdmin}</value>
```

8. Search for `<from-view-id>/app/management/SRManage.jspx`.
9. Add the code in Example 9–5 to the navigation rule.

Example 9–5 Code to Be Added to the navigation-case Tag

```
<navigation-case>
  <from-outcome>SRSiteAdmin</from-outcome>
  <to-view-id>/app/management/SRSiteAdmin.jspx</to-view-id>
</navigation-case>
```

10. Make sure that the code in Example 9–6 is already included.

Example 9–6 Code to Be Added to the managed-bean Tag

```
<managed-bean>
    <managed-bean-name>backing_SRSiteAdmin</managed-bean-name>
<managed-bean-class>oracle.srdemo.view.backing.app.management.SRSiteAdmin</managed
-bean-class>
    <managed-bean-scope>request</managed-bean-scope>
</managed-bean>
```

11. Save the file.
12. In the Applications Navigator, under the UserInterface project, right-click **index.jspx** and select **Run**.

Refer to Chapter 8, "Building a Dashboard Page" for more information about adding a page as a subtab on another page.

Step 5: Using the Site Administration Page for Customization

You will now see how you can customize your application by using the site administration page. You can perform the following tasks on this page:

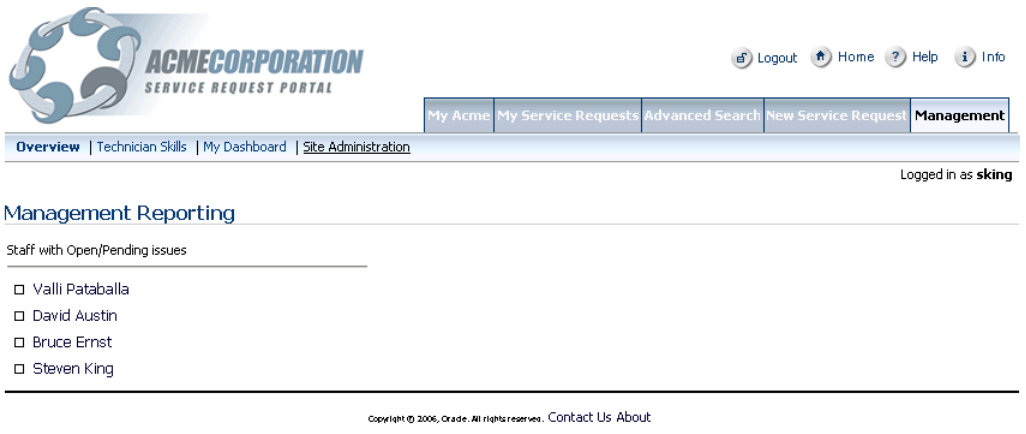
- Changing the Skin for the Application
- Customizing the Login Page

Changing the Skin for the Application

To change the skin for your application, perform the following steps:

1. Log in to the Service Request portal as `sking` with a password of `welcome`.
2. Click the **Management** tab. This page now includes a subtab for your site administration page as shown in Figure 9–8.

Figure 9–8 Management Tab on the Service Request Portal page



Management Reporting

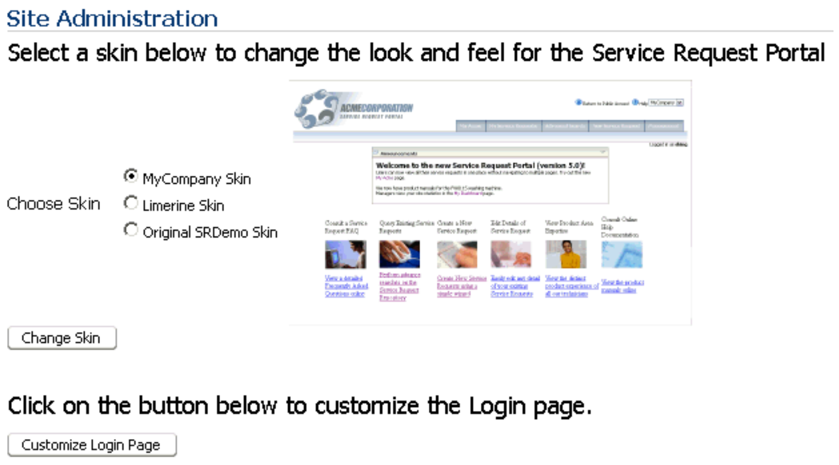
Staff with Open/Pending issues

- Valli Pataballa
- David Austin
- Bruce Ernst
- Steven King

Copyright © 2006, Oracle. All rights reserved. Contact Us About

3. Click the **Site Administration** subtab. This page contains a radio group where the administrator can select the skin to use for the application, as shown in Figure 9–9. The default skin used by the application is `MyCompany Skin`.

Figure 9–9 Site Administration Tab with Customization Options



Site Administration

Select a skin below to change the look and feel for the Service Request Portal

Choose Skin [✓ MyCompany Skin] Limerine Skin Original SRDemo Skin

Click on the button below to customize the Login page.

4. Select **Limerine Skin**. The image updates to give a preview of the skin selected.
5. Click **Change Skin**. The appearance of the page changes to use the `Limerine` skin. This change applies to the complete application.

Customizing the Login Page

If you are a manager, you can customize the login page and edit the text that is displayed to users on this page. To do this, use the Customize Login Page button on the site administration page.

Note: The Customize Login Page button is displayed only to authenticated users who have `customize` privilege on the login page. As you had granted `customize` privilege to the manager role while performing the steps in "Step 4: Editing Authorization for the Login Page" in Chapter 6, "Creating a Login Page", only a manager can customize the login page.

The Customize Login Page button is not displayed to users who have only `view` privilege on the login page.

To customize the login page for your application, perform the following steps:

1. Log in to the Service Request portal as `sking` with a password of `welcome`.
2. Click the **Management** tab. This page now includes a subtab for your site administration page, as shown in Figure 9–8.
3. Click the **Site Administration** subtab. This page provides options to change the skin for your application and to customize the login page, as shown in Figure 9–9.
4. Click the **Customize Login Page** button.
5. On the login page, move your mouse over the text area and click the **actions** button that is displayed. Select **Customize** from the list displayed.
6. Add the text in Example 9–7 to the Rich Text portlet at the top of the page:

Example 9–7 First Rich Text Portlet on the Login Page

```
Please Log In to the Acme Service Request Application
This demonstration application has a number of predefined user accounts
which equate to different roles within the company.
For example:
   sking - a manager
   ahunold - a technician
   dfaviet - a user (customer)
All usernames should be in lowercase with a  password of  "welcome"
Note: If you copy and paste the user IDs from here, be sure to remove any trailing
spaces to ensure that the ID is valid
```

   Use the formatting features of the rich text editor and format this text to appear as shown in Figure 9–10.

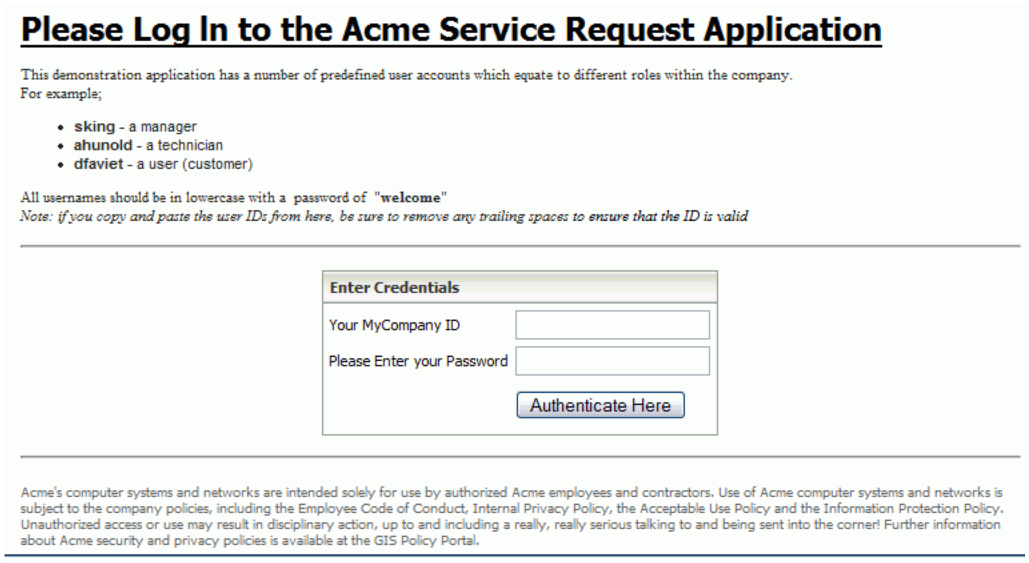
7. Add the text in Example 9–8 to the second Rich Text portlet on the page.

Example 9–8 Second Rich Text Portlet on the Login Page

```
Acme's computer systems and networks are intended solely for use by authorized
Acme employees and contractors. Use of Acme computer systems and networks is
subject to the company policies, including the Employee Code of Conduct, Internal
Privacy Policy, the Acceptable Use Policy and the Information Protection Policy.
Unauthorized access or use may result in disciplinary action, up to and including
a really, really serious talking to and being sent into the corner! Further
information about Acme security and privacy policies is available at the GIS
Policy Portal.to ensure that the ID is valid
```

   Decrease the font size for this text and set the color to a light grey so that the text appears as shown in Figure 9–10.

Figure 9–10 Login Page After Customizing



Please Log In to the Acme Service Request Application

This demonstration application has a number of predefined user accounts which equate to different roles within the company.
For example;

- **sking** - a manager
- **ahunold** - a technician
- **dfaviet** - a user (customer)

All usernames should be in lowercase with a password of "**welcome**"
Note: if you copy and paste the user IDs from here, be sure to remove any trailing spaces to ensure that the ID is valid

Enter Credentials

Your MyCompany ID

Please Enter your Password

Authenticate Here

Acme's computer systems and networks are intended solely for use by authorized Acme employees and contractors. Use of Acme computer systems and networks is subject to the company policies, including the Employee Code of Conduct, Internal Privacy Policy, the Acceptable Use Policy and the Information Protection Policy. Unauthorized access or use may result in disciplinary action, up to and including a really, really serious talking to and being sent into the corner! Further information about Acme security and privacy policies is available at the GIS Policy Portal.

8. Click **Return** to get back to the site administration page.

Summary

In this chapter, you learned how to create a site administration page and add it as a subtab of the Management page. You also learned how to use the site administration page to change the look and feel of the application and to customize the login page.

10

Deploying Your Application

The steps in this chapter will show you how to deploy your application using the Application Server Control Console. To learn how to deploy your application to a production environment, refer to the *Oracle WebCenter Framework Developer's Guide*.

To deploy the SRDemo application to an Oracle Application Server or a standalone OC4J, you must perform the following steps:

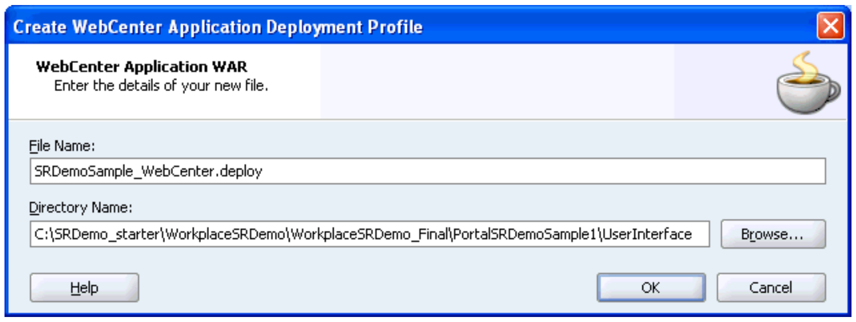
- Step 1: Creating the Generic EAR file
- Step 2: Creating the Targeted EAR File
- Step 3: Setting Up the Users and Roles
- Step 4: Setting Up the Sample Content
- Step 5: Deploying the Application
- Step 6: Deploying the Security Policy
- Step 7: Accessing the Application

Step 1: Creating the Generic EAR file

To create the generic EAR file, perform the following steps:

1. Open your Oracle JDeveloper, go to Applications Navigator, right-click **User Interface**, and select **New**. The New Gallery displays.
2. Under **General**, select **Deployment Profiles** and under **Items** select **WebCenter Application WAR**. The Create WebCenter Application Deployment Profile dialog box displays.
3. In the **File Name** field, enter `SRDemoSample_WebCenter.deploy`, as shown in Figure 10–1, and click **OK**.

Figure 10–1 Create WebCenter Application Deployment Profile



Create WebCenter Application Deployment Profile

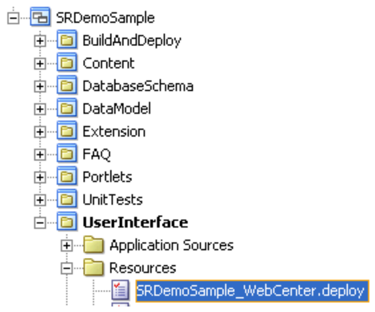
WebCenter Application WAR
Enter the details of your new file.

File Name: SRDemoSample_WebCenter.deploy

Directory Name: C:\SRDemo_starter\WorkplaceSRDemo\WorkplaceSRDemo_Final\PortalSRDemoSample1\UserInterface

4. The deployment profile is created in the **Resources** folder under **User Interface**, as shown in Figure 10–2.

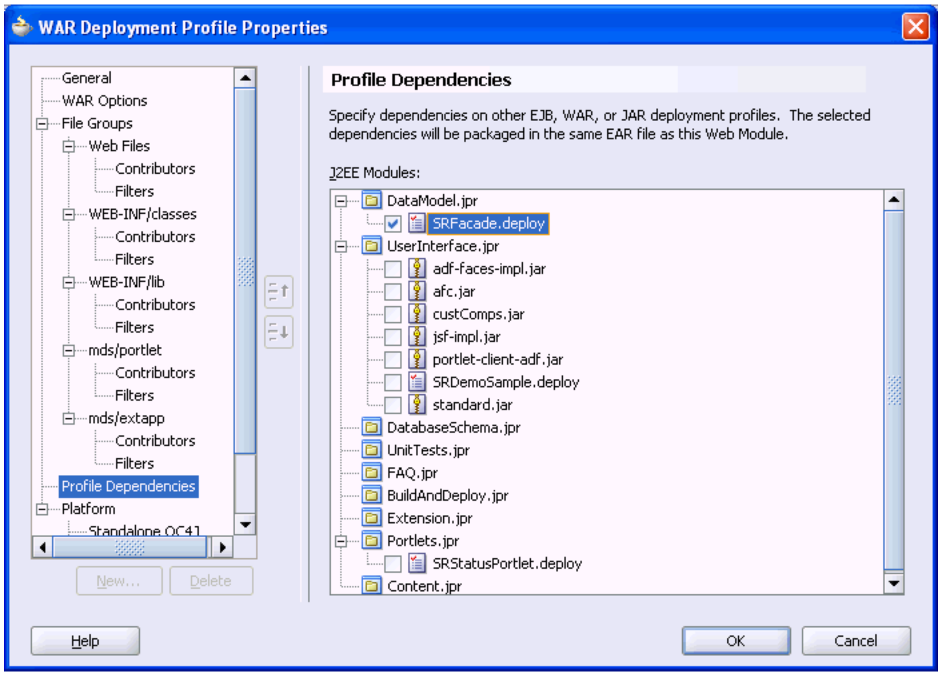
Figure 10–2 Deployment Profile



5. Right-click the deployment profile and select **Properties**. The WAR Deployment Profile Properties dialog box is displayed.

6. Select **Profile Dependencies**, then go to **DataModel.jpr**, and select the **SRFacade.deploy** check box, as shown in Figure 10–3.

Figure 10–3 SRFacade.deploy



Specify dependencies on other EJB, WAR, or JAR deployment profiles. The selected dependencies will be packaged in the same EAR file as this Web Module.

7. Click **OK**.

8. Under **Resources**, right-click **SRDemoSample_WebCenter.deploy** and select **Deploy to EAR file**. This generates the EAR file. The Deployment - Log shows an output similar to Example 10–1.

Example 10–1 Deployment - Log

```
----  Deployment started.  ----    May 29, 2007 1:24:32 PM
Target platform is Standard J2EE.
Exporting portlet metadata and customizations
Wrote EJB JAR file to C:\SRDemo\wc_starter\SRDemoSample_
```

```
WCS\DataModel\deploy\SRFacade.jar
Wrote WAR file to C:\SRDemo\wc_starter\SRDemoSample_
WCS\UserInterface\deploy\SRDemoSample_WebCenter.war
Wrote EAR file to C:\SRDemo\wc_starter\SRDemoSample_
WCS\UserInterface\deploy\SRDemoSample_WebCenter.ear
Elapsed time for deployment:  25 seconds
----  Deployment finished.  ----    May 29, 2007 1:24:57 PM
```

9. Locate the EAR file.
10. Copy it to the machine that has Oracle Application Server running on it. You can do this using a mapped drive on Windows, or by using FTP.

Step 2: Creating the Targeted EAR File

Now, switch to the machine that has your Oracle Application Server installed on it and perform the following steps:

1. Open a command prompt.
2. Run the Predeployment tool to create a targeted EAR file as follows:

```
C:\java\srdemo_oc4j>C:\java\jdev\jdk\bin\java -jar C:\java\srdemo_
oc4j\adfp\lib\portlet-client-deploy.jar -predeploy -source
C:\Temp\SRDemoSample_WebCenter.ear -target C:\Temp\SRDemoSample_WC_target.ear
```

Note: The syntax to create a targeted EAR must be in a single line. It is wrapped in this example for readability.

See chapter titled Deploying Your WebCenter Application" in *Oracle WebCenter Framework Developer's Guide* for explanation of this syntax.

Where the source file is the generic EAR file created earlier.

You will be prompted to specify various settings while you run the Predeployment tool. The output will look like Example 10–2 (user defined settings in **bold**):

Example 10–2 Output of the Predeployment Tool

```
Processing Arguments
Run Mode : 1
Source : C:\Temp\SRDemoSample_WebCenter.ear
Target : C:\Temp\SRDemoSample_WC_target.ear
Deployed App : null
Mapping File : null
Deployed App : null
Cleaning up C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\predeploy\
May 16, 2007 12:25:33 PM oracle.adf.share.config.ADFConfigFactory findOrCreateADF
Config
INFO: oracle.adf.share.config.ADFConfigFactory No META-INF/adf-config.xml found
Processing source EAR file
Source EAR file processed
Processing adf-config.xml
adf-config.xml processed
Processing connections.xml
connections.xml processed

Creating a new Deployment Profile for : C:\Temp\SRDemoSample_WebCenter.ear
```

```
Development MDS Repository Path : ../../mds/;../../
Enter new MDS Repository Path :
C:\java\srdemo_oc4j\mds\SRDemo

Producer : OmniPortletProducer_11743434494566c52fb80-0111-1000-8001-a9fe02025106

Current Service URL :  http://pdtsdemo1.us.oracle.com:8888/portalTools/omniPortl
et/providers/omniPortlet
Current Proxy URL   :
Current Proxy Port   : 0
Do you want to modify this connection? (Y/N [default=N]) :
N

Validating producer OmniPortletProducer_
11743434494566c52fb80-0111-1000-8001-a9fe02025106
.
.
.
Do you wish to save this new Deployment Profile (Y/N [default=N]) :
y
Enter a name for this Deployment Profile (e.g. 'Production') :
srdemo_deploy
Enter the path for the Deployment Profile XML file (e.g. 'C:\profile.xml'):
C:\temp\srdemo_deploy.xml
Saving to C:\temp\srdemo_deploy.xml
Cleaning up C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\predeploy\
Extracting View Documents, MDS and connections.xml from source EAR file
Examining contents of SRDemoSample_WebCenter.war
Moving C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\predeploy\tmppkg\views\SRDemoSample_
WebCenter\WEB-INF to
C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\predeploy\view\UserInterface\public_
html\WEB-INF
.
.
.
MDS extracted from source EAR file
Creating target EAR file
Processing source EAR file
.
.
.
Source EAR file processed
Processing adf-config.xml
adf/META-INF/adf-config.xml [Recreated]
adf-config.xml processed
Processing connections.xml
Validating producer OmniPortletProducer_
11743434494566c52fb80-0111-1000-8001-a9fe02025106
.
.
.
adf/META-INF/connections.xml [Recreated]
connections.xml processed
Processing SRDemoSample_WebCenter.war
WEB-INF/lib/META-INF/adf-config.xml [Recreated in WAR file]
WEB-INF/classes/META-INF/connections.xml [Recreated in WAR file]
WEB-INF/lib/META-INF/connections.xml [Recreated in WAR file]
SRDemoSample_WebCenter.war [Recreated]
WAR file processed
```

```
Creating target EAR file
WAR file processed
source MDS Path (temp) : C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\predeploy\mds
Production MDS Path : C:\java\srdemo_oc4j\mds\SRDemo
Connections.xml Path :
C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\predeploy\connections.xml.new
Export ID : /export
May 16, 2007 12:25:58 PM oracle.mds
NOTIFICATION:
"oracle.portlet.client.persistenceimpl.mds.ImportListener"::"preOperation" is
being invoked.
.
.
.
NOTIFICATION: Transferring the document /UserInterface/public_html/app/staff/SRS
taffSearch.jspx.
NOTIFICATION: import is completed. Total number of documents successfully proces
sed : 24, total number of documents failed : 0.
Moving C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\predeploy\SRDemoSample_WC_target.ear t
o C:\Temp\SRDemoSample_WC_target.ear
Target EAR C:\Temp\SRDemoSample_WC_target.ear created
Cleaning up C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\predeploy\
```

Note: in Example 10–2, existing settings for all the portlet producers remain unchanged. You can change them by specifying `Y` and then providing the appropriate path to the new producer location.

Step 3: Setting Up the Users and Roles

Since the SRDemo users and roles will not yet exist on Oracle Application Server to which you want to deploy the application, you must update the global `system-jazn-data.xml` file for the application server or standalone OC4J by running the JAZN Migration Tool in realm mode.

Note: You have already performed this step in the development environment, but since you may not have Oracle JDeveloper installed on the system that hosts your Oracle Application Server, these steps will show you how to do it by calling the JAZN Migration tool directly.

To add the SRDemo users and roles, perform the following steps:

1. Locate the `system-jazn-data-starter.xml` file in the starter ZIP file. This contains the users and roles.
2. Backup your OC4J's `system-jazn-data.xml` file, located in *`ORACLE_HOME`*`\j2ee\home\config\system-jazn-data.xml`.
3. Open a command prompt.
4. Set the classpath as follows:

   ```
   set CLASSPATH = ORACLE_HOME/j2ee/home/jazn.jar;ORACLE_
   HOME/BC4J/lib/adfshare.jar
   ```

   For example, if your ORACLE_HOME is `C:\java\srdemo_oc4j`, set the classpath as follows:

```
set CLASSPATH = C:\java\srdemo_oc4j/j2ee/home/jazn.jar; C:\java\srdemo_
oc4j/BC4J/lib/adfshare.jar
```

5. Run the JAZN Migration tool as follows (This assumes the same Oracle home location again.)

```
java oracle.security.jazn.tools.JAZNMigrationTool -sr jazn.com -dr jazn.com -st
xml -dt xml -sf C:\Temp\system-jazn-data-starter.xml -df C:\java\srdemo_
oc4j\j2ee\home\config\system-jazn-data.xml -m realm
```

Step 4: Setting Up the Sample Content

You must set up the sample content on the system that hosts your Oracle Application Server in the same way you did this for the development environment. To do this, extract the content of `srcontentrepository.zip` from `C:\SDemo_App_Download to C:\srdemo`.

Note: If you are using UNIX, you will need to update your `connections.xml` file with the path to the content repository. Open the `connections.xml` file and search for `C:\srdemo\SRContentRepository`, and update the path to point to the directory where you extracted the sample content.

If you are using Windows and you chose not to extract the content to the suggested directory, you will also need to update your `connections.xml` file with the path you chose.

Step 5: Deploying the Application

Now that you have generated the targeted EAR file, you are ready to deploy it. You do this using Application Server Control Console as follows:

1. To access Application Server Control Console, navigate to the following URL: `http://`*`<host_name>.<domain>:<port>`*`/em`.

   For example, `http://test.acme.com:8888/em`.

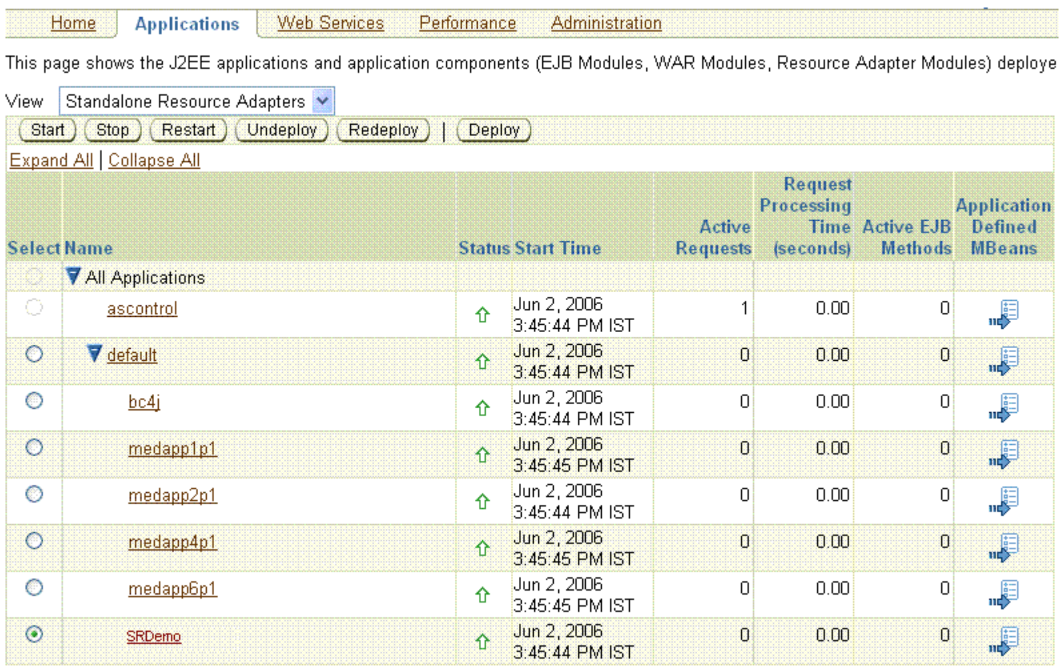
   To find the exact URL for your Application Server Control Console, look at `readme.txt`. After installation, this text file is saved to the following Oracle Application Server location:

   On UNIX: *`ORACLE_HOME`*`/install/readme.txt`

   On Windows: *`ORACLE_HOME`*`\install\readme.txt`

2. Log in to the Application Server Control Console. The Cluster Topology page is displayed.
3. On the Cluster Topology page, click the link to your application server.
4. Select the **Applications** tab (Figure 10–4) and click **Deploy**. The Deploy: Select Archive page is displayed.

Figure 10–4 Applications Tab of Application Server Control Console



This page shows the J2EE applications and application components (EJB Modules, WAR Modules, Resource Adapter Modules) deploye

View Standalone Resource Adapters

Select	Name	Status	Start Time	Active Requests	Request Processing Time (seconds)	Active EJB Methods	Application Defined MBeans
	All Applications						
○	ascontrol	⇧	Jun 2, 2006 3:45:44 PM IST	1	0.00	0	
○	default	⇧	Jun 2, 2006 3:45:44 PM IST	0	0.00	0	
○	bc4j	⇧	Jun 2, 2006 3:45:44 PM IST	0	0.00	0	
○	medapp1p1	⇧	Jun 2, 2006 3:45:45 PM IST	0	0.00	0	
○	medapp2p1	⇧	Jun 2, 2006 3:45:44 PM IST	0	0.00	0	
○	medapp4p1	⇧	Jun 2, 2006 3:45:45 PM IST	0	0.00	0	
○	medapp6p1	⇧	Jun 2, 2006 3:45:45 PM IST	0	0.00	0	
●	SRDemo	⇧	Jun 2, 2006 3:45:44 PM IST	0	0.00	0	

5. Select the **Archive is present on local host** option and browse to the location of the EAR file. Then, click **Next**.

6. In the Deploy: Application Attributes page, enter the context root of your Web application as the application name or what you configured while creating the deployment profile.

7. Deployment Settings page is displayed. If required, then make the changes in this page and click **Deploy**.

8. The deployment confirmation page is displayed with output like Figure 10–3.

Example 10–3 Deployment Confirmation Output

```
[May 16, 2007 5:05:11 PM] Application Deployer for SRDemo STARTS.
[May 16, 2007 5:05:11 PM] Copy the archive to C:\java\srdemo_
oc4j\j2ee\home\applications\SRDemo.ear
[May 16, 2007 5:05:12 PM] Initialize C:\java\srdemo_
oc4j\j2ee\home\applications\SRDemo.ear begins...
[May 16, 2007 5:05:12 PM] Unpacking SRDemo.ear
[May 16, 2007 5:05:15 PM] Done unpacking SRDemo.ear
[May 16, 2007 5:05:15 PM] Unpacking SRDemoSample_WebCenter.war
[May 16, 2007 5:05:25 PM] Done unpacking SRDemoSample_WebCenter.war
[May 16, 2007 5:05:26 PM] Initialize C:\java\srdemo_
oc4j\j2ee\home\applications\SRDemo.ear ends...
[May 16, 2007 5:05:26 PM] Starting application : SRDemo
[May 16, 2007 5:05:26 PM] Initializing ClassLoader(s)
[May 16, 2007 5:05:26 PM] Initializing EJB container
[May 16, 2007 5:05:26 PM] Loading connector(s)
[May 16, 2007 5:05:27 PM] Starting up resource adapters
[May 16, 2007 5:05:27 PM] Processing EJB module: SRFacade.jar
[May 16, 2007 5:05:29 PM] Initializing EJB sessions
[May 16, 2007 5:05:29 PM] Committing ClassLoader(s)
[May 16, 2007 5:05:29 PM] Initialize SRDemoSample_WebCenter begins...
[May 16, 2007 5:05:29 PM] Initialize SRDemoSample_WebCenter ends...
[May 16, 2007 5:05:29 PM] Started application : SRDemo
```

```
[May 16, 2007 5:05:29 PM] Binding web application(s) to site default-web-site
begins...
[May 16, 2007 5:05:29 PM] Binding SRDemoSample_WebCenter web-module for
application SRDemo to site default-web-site under context root SRDemo
[May 16, 2007 5:06:54 PM] Initializing Servlet: javax.faces.webapp.FacesServlet
for web application SRDemoSample_WebCenter
[May 16, 2007 5:06:55 PM] Initializing
Servlet:oracle.adfinternal.view.faces.renderkit.html.portlet.ADFPortletServlet for
web application SRDemoSample_WebCenter
[May 16, 2007 5:06:55 PM] Binding web application(s) to site default-web-site
ends...
[May 16, 2007 5:06:55 PM] Application Deployer for SRDemo COMPLETES. Operation
time: 103484 msecs
```

9. Go to the **Applications** tab and select your application. Then, click **Administration**.
10. Under **Administration Tasks**, click **Configure Class Loading**.
11. De-select **Apache.commons.logging**.

The application is now deployed, but you cannot access it yet. You must first deploy the security policies.

Note: For this example, we expect that the sample still resides in the same database. The targeted EAR file will be configured to use the database connection that you configured in the development environment. If this connection does not exist, you may need to use Application Server Control Console to create a new database connection using the same name that you used in the development environment. See the installation file section titled "Running the Completed Demo." for more information about running the database scripts.

Step 6: Deploying the Security Policy

The security policy is bundled in the application's EAR file in the `app-jazn-data.xml` file. To deploy the policies for your application, perform the following steps:

1. Locate the `app-jazn-data.xml` file. If your ORACLE_HOME is `C:\java\srdemo_oc4j` , then this file should be located at `C:\java\srdemo_oc4j\j2ee\home\applications\SRDemo\adf\META-INF\app-jazn-data.xml`.
2. Open a command prompt.
3. Set the classpath as follows:

   ```
   set CLASSPATH = ORACLE_HOME/j2ee/home/jazn.jar;ORACLE_
   HOME/BC4J/lib/adfshare.jar
   ```

   For example, if your ORACLE_HOME is `C:\java\srdemo_oc4j`, set the classpath as follows:

   ```
   set CLASSPATH = C:\java\srdemo_oc4j/j2ee/home/jazn.jar; C:\java\srdemo_
   oc4j/BC4J/lib/adfshare.jar
   ```

4. Run the JAZN Migration tool as follows (This assumes that `C:\java\srdemo_oc4j` is the Oracle home location.):

```
java oracle.security.jazn.tools.JAZNMigrationTool -sr jazn.com -dr jazn.com
-st xml -dt xml -sf C:\java\srdemo_
oc4j\j2ee\home\applications\SRDemo\adf\META-INF\app-jazn-data.xml -df
C:\java\srdemo_oc4j\j2ee\home\config\system-jazn-data.xml -m policy
```

5. Restart OC4J.

This migrates the policy information. You are now ready to run the application.

Step 7: Accessing the Application

You can now access the SRDemo. To do this, perform the following steps:

1. Open a browser and navigate to

   ```
   http://appserver_host_name:appserver_port_
   number/SRDemo/faces/app/SRWelcome.jspx
   ```

2. Log in to see the SRDemo.

Summary

This chapter showed you how to take your application and deploy it. You should now be able to run your application and view the pages and components you created throughout this manual. You can also compare your application to the completed version provided to you in the sample files. To do so, follow the instructions to run the completed demo in the install.html file.

Index

A

ADF table
  wiring a SelectOneChoice, 7-11
  wiring portlets, 7-14
ADF, See Application Development Framework, 1-6
application
  deploying, 10-1
  deploying the security policy, 10-8
  deploying using the JAZN migration tool, 10-6
  setting up users and roles, 10-5
  using Application Server Control Console, 10-6
  using the predeployment tool, 10-3
Application Development Framework (ADF), 1-6
Application Service
  connecting, 2-3
authorization
  editing for login page, 6-15

C

component
  adding a JavaServer Faces drop-down component, 7-12
  adding a search component to a page, 8-11
  adding a SelectOneChoice, 8-10
  adding an ADF table to a page, 7-22
  adding security, 7-24
  wiring a SelectOneChoice to an ADF table, 7-11
  wiring components together on a page, 8-15
connection
  creating a connection to your Application Server, 2-3
Content DB
  about, 1-5
content repository
  configuring JCR data controls, 4-3
  displaying a file in place, 5-4
  setting up, 4-1
customization
  site administration page, 9-10
customization components
  adding to SRMyAcme.jspx page, 7-5
  using to create a page layout, 8-4

D

dashboard page
  about, 8-1
  creating, 8-2
data controls
  configuring, 4-3
deployment
  creating the generic EAR file, 10-1
  creating the targeted EAR file, 10-3
  using the Application Server Control Console, 10-6

E

example
  description, 1-7

G

generic EAR file
  creating, 10-1

I

interactivity
  adding to a page, 7-14

J

JavaServer Faces drop-down component
  adding to a page, 7-12
JAZN migration tool
  deploying your application, 10-6
JCR adapters, 1-4
JCR data controls, 1-4
JSR 168 portlet
  adding logic, 2-10
  creating, 2-4, 2-5
  deploying to an Application Server, 2-11
  registering the producer, 2-14

L

layout components
  adding to SRMyAcme.jspx page, 7-5
lifecycle support

about, 1-5
login page
adding Rich Text portlet, 6-14
authentication settings, 6-16
configuring the application, 6-16
creating, 6-1, 6-2
creating a login error page, 6-11
customizing, 9-12
editing authorization, 6-15
getLoginFormBlock getter method, 6-7
getloginScriptBlock getter method, 6-6
getloginStyleBlock getter method, 6-5
Oracle ADF Security Wizard, 6-16

O

ObjectSpacer
adding, 7-9
OmniPortlet
adding instances to a page, 8-9
defining with a pie chart layout, 8-13
defining with a piechart layout, 8-12
defining with a SQL data source, 7-13
defining with a tabular layout, 8-11
defining with an HTML layout, 8-13, 8-14
description, 1-3
registering the producer, 7-4
Oracle ADF Security Wizard, 6-16
Oracle Content Database
about, 1-5
Oracle JDeveloper
and ADF, 1-6
Oracle WebCenter Suite
contents, 1-3

P

page
about a dashboard page, 8-1
adding a PDK-Java portlet, 7-14
adding an ADF table, 7-22
adding interactivity, 7-14
adding OmniPortlet instances, 8-9
adding security, 7-23
creating, 7-2
creating a dashboard, 8-2
creating a site administration page, 9-1
using customization components, 8-4
wiring content on a page, 8-15
page layout
creating, 8-4
PanelCustomizable
adding, 7-5
PanelHorizontal
adding, 7-10
using, 7-8
PDK-Java portlet
adding logic, 2-21
adding to a page, 7-14
building, 2-16
deploying to an Application Server, 2-26
portlets
adding a PDK-Java portlet to a page, 7-14
adding a Web Clipping portlet, 7-18
adding logic to a JSR 168 portlet, 2-10
adding logic to a PDK-Java portlet, 2-21
adding OmniPortlet instances to a page, 8-9
building a PDK-Java portlet, 2-16
creating a JSR 168 portlet, 2-4, 2-5
creating a PDK-Java portlet and producer, 2-16
creating a project, 2-1
defining an OmniPortlet, 7-13
defining an OmniPortlet with a pie chart layout, 8-12, 8-13
defining an OmniPortlet with a tabular layout, 8-11
defining an OmniPortlet with an HTML layout, 8-13, 8-14
deploying a JSR 168 portlet, 2-11
deploying a PDK-Java portlet, 2-26
registering a JSR 168 portlet producer, 2-14
wiring to an ADF table, 7-14
predeployment tool
using, 10-3
producer
creating a PDK-Java portlet producer, 2-16
registering, 2-14
registering a Web Clipping producer, 7-19
registering the OmniPortlet producer, 7-4

R

rich text component
adding to welcome page, 5-1
Rich Text portlet
description, 1-4
roles
setting up for your application, 10-5

S

schedule on-site services portlet
creating, 7-18
search
adding a search component to a page, 8-11
security
adding to a page, 7-23
applying to components, 7-24
security policy
deploying, 10-8
SelectItem
adding, 7-7
SelectOneChoice
adding, 7-6
adding to a page, 8-10
wiring to an ADF table, 7-11
Service Request Demo
description, 1-7
ShowDetailFrame
adding, 7-6

ShowDetailItem
  adding, 7-10
ShowOneTab
  adding, 7-9
site administration page
  changing skin, 9-10
  creating, 9-1
  customizing login page, 9-12
skin selector
  adding, 9-3
skins
  configuring your application, 3-2
  registering, 3-1
  selecting, 9-10
  using, 9-10
SRLogin.jspx
  creating, 6-1
SRMyAcme.jspx
  adding customization components, 7-5
  adding Web Clipping portlet, 7-18
SRSiteAdmin.jspx
  creating, 9-2

T

targeted EAR file
  creating, 10-3

U

URLs
  specifying for Web Clipping, 7-20
users and roles
  setting up for your application, 10-5

W

Web Clipping portlet
  adding, 7-19
  creating, 7-18
  description, 1-4
  registering, 7-19
  selecting the Web page, 7-19
Web content
  specifying, 7-20
WebCenter applications
  WAR, 1-5
WebCenter Framework
  description, 1-3
WebCenter Services
  about, 1-5
web.xml
  updating, 6-18
welcome page
  adding global navigation links, 5-8
  adding rich text component to, 5-1
  creating, 5-1
  displaying a file in place, 5-4
  making publicly accessible, 5-11