Oracle WebCenter
Hands-On Practices

A Practical Introduction
to Oracle WebCenter
# Table of Contents

Before You Start ............................................................................................................. 3  
Estimated timings for the practices ............................................................................. 3  
Accessing the hands-on setup files ............................................................................. 3  
Practice 1: Consuming Portlets in a JSF Application ..................................................... 4  
Create a new application and projects in JDeveloper ................................................. 5  
Create a JSF Page ........................................................................................................ 6  
Add an ADF read-only table to the Tours page .......................................................... 8  
Run the page ............................................................................................................. 14  
Add a portlet to the Tours page ................................................................................. 14  
Configure the portlet ................................................................................................. 20  
Practice 2: Adding Customization to Your Application ............................................... 25  
Add a showDetailFrame component to the page ...................................................... 26  
Run and test the page ................................................................................................ 27  
Enable customization ................................................................................................ 29  
Update the customizable components’ properties ..................................................... 31  
Run and test your page .............................................................................................. 34  
Add tabs to the Tours page ....................................................................................... 36  
Practice 3: Linking the Components ............................................................................. 39  
Look at the predefined page parameters ................................................................... 40  
Set the default value for OmniPortlet1_1_Param1 ................................................... 42  
Customize the portlet to use the parameter ............................................................... 43  
Test that the components are synchronized .............................................................. 44  
Automate the component synchronization ................................................................ 47  
Practice 4: Integrating Content from a JCR 1.0 Repository ......................................... 50  
Create a file system data control .............................................................................. 51  
Create an ADF Faces tree based on the data control ................................................ 52  
Make the file names hyperlinks ................................................................................ 58  
Run the page ............................................................................................................. 70  
Practice 5: Building and Deploying Standards-Based Portlets..................................... 73  
Create the ListTours portlet ...................................................................................... 74  
Deploy the portlet ..................................................................................................... 79  
Consume the portlet .................................................................................................. 85  
Add the Parameter Form portlet ............................................................................... 89  
Add logic to the ListTours portlet .......................................................................... 94  
Link the portlets together ........................................................................................ 101  
Practice 6: Portletizing an ADF Faces Application .................................................... 106  
Create a page to portletize ........................................................................................ 107  
Portletize the page ................................................................................................... 111  
Deploy the portlet ................................................................................................... 115  
Consume the portlet ................................................................................................. 118
Before You Start
This book contains six practices, which progressively build the Veeva application.

Estimated timings for the practices
These timings are only approximate, and will vary greatly, depending on your prior experience in using JDeveloper and ADF Faces.

<table>
<thead>
<tr>
<th>Practice</th>
<th>Description</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Consuming Portlets in a JSF Application</td>
<td>45 min</td>
</tr>
<tr>
<td>2</td>
<td>Adding Customization to Your Application</td>
<td>40 min</td>
</tr>
<tr>
<td>3</td>
<td>Linking the Components</td>
<td>30 min</td>
</tr>
<tr>
<td>4</td>
<td>Integrating Content from a JCR 1.0 Repository</td>
<td>60 min</td>
</tr>
<tr>
<td>5</td>
<td>Building and Deploying Standards-Based Portlets</td>
<td>55 min</td>
</tr>
<tr>
<td>6</td>
<td>Portletizing an ADF Faces Application</td>
<td>30 min</td>
</tr>
</tbody>
</table>

Setting up the Veeva application files
You can download a zip file of all the files and instructions that you need to set up your own Veeva environment:

1) Start up JDeveloper. If this is the first time you’ve used this JDeveloper installation, you need to install the preconfigured OC4J. To do this, click the traffic light icon (top right). When prompted to install OC4J, click OK.
2) Go to the WebCenter home page (webcenter.oracle.com)
3) Click the Demonstrations and Samples link.
4) On the Demos and Solutions page, click the Veeva Vacations Setup Materials link.
5) Follow the steps in the readme file to set up Veeva. Specifically:
   a) Unzip veeva_setup.zip. This creates the VeevaSetup directory.
   b) Unzip VeevaSetup\Veev.zip into the top level of your C: drive.
   c) Copy all the files from the VeevaSetup\CopyToOC4J folder to the default Web application folder of your preconfigured OC4J; for example, C:\jdev\jdev\extensions\oracle.adfp.seededoc4j.10.1.3.2.0\j2ee\home\default-web-app
Practice 1: Consuming Portlets in a JSF Application

In this practice, you will do the following:

- Create a JSP page called Tours.
- Add an ADF Faces table that is based on a database data control.
- Add an out-of-the-box portlet to the page.
- Configure the portlet to read and display tour details and images from a CSV file.

This is how the Tours page will look after you complete this practice:
Create a new application and projects in JDeveloper

1) Start Oracle JDeveloper by double-clicking the icon on your Windows desktop.

2) Right-click the Applications node in the Applications Navigator, and then select New Application.

3) In the Create Application window, enter Veeva as the name of the new application. Use the Application Template list to select “WebCenter Application [Portlet, Content Repository, JSF].” Click OK.
Create a JSF Page

1) From the Applications Navigator, right-click ViewController, and select New from the context menu.

2) In the New Gallery, select Web Tier > JSF, and then select JSF JSP in the Items pane.

3) Click OK. The Create JSF JSP Wizard opens.

4) On the Welcome page, click Next.
5) Name the file Tours.jspx, and then select **JSP Document**, which is an XML representation of a JSP page. Click **Next**.

4) On the Component Binding page, choose **Automatically Expose UI Components in a New Managed Bean** so that JDeveloper will create a backing bean for the page. Click **Next**.
5) On the Tag Libraries page, select the JSP tag libraries to use in the JSP file. Ensure you have the libraries shown below selected:

![Oracle WebCenter: A Practical Introduction](image)

6) Click **Finish**. Notice that Tours.jspx appears in the Visual Editor as a blank page. The broken red box indicates the form on the page.

**Add an ADF read-only table to the Tours page**

**Define a Database Connection**

1) Click the Connections Navigator tab.

![Oracle WebCenter: A Practical Introduction](image)

2) Right-click **Database**, and then select **New Database Connection**.
3) Enter the Connection Name (**Vacations**), and then select Oracle (**JDBC**) as the Connection Type. Click **Next**.

4) In the Authentication page, enter the Username (**veeva**) and Password (**oracle1**). Do not specify a role. Select **Deploy Password**. Click **Next**.
5) In the Connection page, select **thin** driver, and enter the host name, port, and SID as provided by your instructor. Click **Next**.

6) Click **Test Connection** and look for the Success! message.

7) Click **Finish**.
Create an ADF read-only table

1) Expand the **Database** node; then expand **Vacations > VEEVA > Tables** to find the **COUNTRIES** table.

![Database Hierarchy]

2) Click and drag **COUNTRIES**, and drop it on the **Tours** page in the Visual Editor.
3) When the Create context menu pops up, select **Tables > ADF Read-only Table**.

The Edit Table Columns dialog opens.
4) Make the following changes in this dialog:
   a) Delete the **CountryId** and **Currency** columns.
   b) Select the “**Enable selection**” and “**Enable sorting**” options.

c) Click **OK**. The table appears on the page.
Run the page

1) Right-click Tours.jspx in the Visual Editor, and then select Run from the context menu.

2) Notice the Select radio button and the sortable columns.

Add a portlet to the Tours page

Up to this point, you've been building an ADF Faces application. Now, you will take advantage of WebCenter functionality by adding a portlet to your application.

Start the preconfigured OC4J

Before adding a portlet to the JSF page, you need to start an OC4J, which is Oracle's J2EE runtime. WebCenter includes a preconfigured OC4J with portlets such as Web Clipping, OmniPortlet, and Rich Text Portlet.
1) Click the green traffic light icon on the right side of the toolbar.

2) Look for the “Oracle Containers for J2EE (10.1.3.1.0) initialized” message in the OC4J log.

3) While OC4J is initializing, take a look at the WebCenter Preconfigured OC4J readme page. Select Help > WebCenter Preconfigured OC4J Readme. This page contains helpful information and links.

4) Click the index page link in the Index Page section of the readme page to invoke the home page of the preconfigured OC4J. This page enables you to test OC4J and find links to preconfigured portlet producers.

10.1.3.2.0 directory on Windows. On UNIX platforms, it installs in the $HOME/jdevhome/extensions/oracle.adfp.seedocs4j.10.1.3.2.0 directory.

If you have already set the $JDEV_USER_DIR environment variable, users who share the same Oracle JDeveloper installation but have different $JDEV_USER_DIR directories will have an individually preconfigured OC4J installation. In this case, the preconfigured OC4J installs in the $JDEV_USER_DIR/extensions/oracle.adfp.seedocs4j.10.1.3.2.0 directory.

Index Page

After you start the preconfigured OC4J, confirm that it has started by waiting for the Oracle Containers for J2EE 10g (10.1.3.1.0) initialized message in the Log window in Oracle JDeveloper.

Once OC4J starts, bring up the index page to test the OC4J. The index page contains a list of links to the preconfigured portlet producers. You can test each of the portlet producers by clicking the links.

Default Environment Settings
5) Click the **OmniPortlet Producer** link.

### WebCenter Preconfigured OC4J

<table>
<thead>
<tr>
<th>OC4J 10.1.3.1 Preconfigured for Oracle WebCenter 10.1.3.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Congratulations!</td>
</tr>
</tbody>
</table>

Your server is up and running. This server is preconfigured for Oracle WebCenter, and contains the following components:

- Stand-alone OC4J 10.1.3.1.0
- Oracle Portlet Container (JSR 188/WSRP) 10.1.3.2.8
- PortalTools 10.1.3.2.0
  - OmniPortlet
  - Web Clipping
- Rich Text Portlet 10.1.3.2.0
- JSF-Portlet Bridge 10.1.3.2.0
- ADF Runtime Libraries 10.1.3.2.0

### Preconfigured Portlet Producers

This server contains the following preconfigured portlet producers:

#### WSRP Portlet Producers

- Rich Text Portlet Producer
- Sample Portlets

#### PortalTools Portlet Producers

- PortalTools Welcome Page

**Note**: An OmniPortlet enables developers to quickly and easily publish data from various data sources in a variety of layouts by using a Web-based wizard. The OmniPortlet can be based on almost any kind of data source such as a spreadsheet, XML, Web services, RSS feed, database, or even application data from an existing Web page. This link takes you to the OmniPortlet home page.

6) Copy the URL in the address field to the clipboard. You will use this to register the OmniPortlet.

7) Return to JDeveloper.

**Register a new producer**

The next step is registering the OmniPortlet producer so that you can add an OmniPortlet to the Tours page.

**Note**: The OC4J must be running for a successful registration.
1) Click the Applications Navigator tab (top right).

2) Right-click the View Controller project in the navigator, and select New from the context menu.

3) In the New Gallery, select Web Tier > Portlets. In the Items pane, select “Oracle PDK-Java Producer Registration”.

Notice there are two types of producer registrations listed:
- PDK-Java registration, for portlets that use Oracle's proprietary Java portlet API
- WSRP registration, for standards-based portlets

4) Click OK. The PDK Portlet Producer Registration Wizard opens.

5) On the Welcome page, click Next.
6) Name the producer **OmniProducer**, and then click **Next**.

7) In the **URL Endpoint** field, paste the URL that you copied from the OmniPortlet home page.

8) Click **Finish**.

9) Click **OK** to acknowledge the successful registration of the OmniProducer.

---

**Add the portlet to a page**

1) Click the **Tours.jspx** tab to return to the Tours page in the Visual Editor.

2) Click the **Components** tab (top right) and select **OmniProducer** from the list of component palettes.
3) Click and drag OmniPortlet, and drop it on the form (that is, inside the red box) in the Visual Editor.

To configure the portlet, you need to run the page. Before running the page, it’s a good idea to stop the embedded OC4J.

4) Click the red square in the Embedded OC4J Server log.

5) You need to run the page to configure the portlet. Right-click Tours.jspx in the Visual Editor, and then select Run from the context menu.
Configure the portlet

1) Click **Define** to invoke a wizard that will step you through the configuration process.

2) Select **Spreadsheet** for the type of data. Click **Next**.
3) On the **Source** page, enter the following as the CSV URL, and then click **Next**:

```
http://localhost:6688/tours.csv
```

4) Make the following changes on the **Filter** page:

- Specify a condition of `country_id` *Equals* 1002.
- Limit the result to 2.

Then click **Next**.

5) On the **View** page, make the following changes:

- Change the title to **Veeva Vacations**.
- Select **HTML** as the layout style.
Then click Next.

6) On the Layout page, make the following changes:

   a) In the Quick Start section, with Clear Fields selected, click Apply.

   b) In Notepad, open VeevaSetup\omniLayout.txt, which contains HTML that you can use to control the layout of the data.
c) Copy the `<table>` tag from Notepad, and paste it in the **Non-Repeating Heading Section**.

d) Copy the entire body section (except for the text “body section”) from Notepad, and paste it in the **Repeating Section**.

e) Copy the `</table>` tag from Notepad, and paste it in the **Non-Repeating Footer Section**.

7) Click **Finish**.
Oracle WebCenter: A Practical Introduction

You should see the Countries table at the top of the page and the Veeva Vacations portlet with two tour descriptions and images.

Congratulations! You have successfully completed the first practice!
Practice 2: Adding Customization to Your Application

In this practice, you will do the following:

- Add a border and header (“chrome”) around the ADF table component by using a showDetailFrame component.
- Add customization to a group of components by using a panelCustomizable component.
- Add tabs to the Tours page.

This is how the Tours page will look after you complete this practice:
Add a showDetailFrame component to the page

1) From the list of component palettes, select Customizable Components Core. Customizable Core components are part of WebCenter Framework.

2) Click and drag ShowDetailFrame and drop it anywhere inside the form on the page.

3) In the Structure pane (bottom left), select af:table – table1 and drop it on the cust:showDetailFrame.
You can see in both the Structure pane and the Visual Editor that the portlet is now positioned above the showDetailFrame, and the table appears inside the showDetailFrame.

4) In the Structure pane, to resequence the components, drag cust:ShowDetailFrame and drop it just below h:form. The Structure pane should look similar to the one pictured below.

Run and test the page

1) Stop the embedded OC4J by clicking the red square in the Embedded OC4J Server log.
2) Right-click the Tours page in the Visual Editor and select Run.

3) Notice that the ADF Faces table now looks similar the portlet because it is contained within a showDetailFrame.

4) Click the control in the top left of the showDetailFrame to collapse it. Click a second time to expand it.
Enable customization
To expose customization for the ADF Faces table component, you need to wrap its `showDetailFrame` in a `PanelCustomizable` component.

1) In JDeveloper, click and drag `PanelCustomizable` from the Customizable Core Components palette and drop it on the form in the Structure pane.

2) In the Structure pane, drag the `showDetailFrame` and drop it in the `PanelCustomizable` component.
3) Now move adfp:portlet into the panelCustomizable component.

![Diagram of panelCustomizable component]

**Note:** In addition to exposing customization for components, panelCustomizable gives you more control over the layout of components. This is the reason for moving the portlet to the panelCustomizable component.

4) Compare the structure inside your panelCustomizable component with the screen shot below. If necessary, use drag and drop to resequence the components in your panelCustomizable.

![Diagram of panelCustomizable component structure]

5) Stop the embedded OC4J by clicking the red square in the Embedded OC4J Server log.

![Embedded OC4J Server log]

6) Run the page to observe the result of your changes. Right-click Tours.jspx in the Visual Editor, and then select Run from the context menu.
7) Notice that wrapping the showDetailFrame in a panelCustomizable component has exposed the control with actions (top right) to move the showDetailFrame up or down and to maximize it.

8) Maximize the showDetailFrame, then minimize it. Move the component so that it’s below the portlet, and then move it back again.

**Update the customizable components’ properties**

Now that the page is customizable, there are some other enhancements to make, such as:

- Arranging the components horizontally
- Changing the title of the showDetailFrame
- Changing the header color of both components
- Adding white space to visually separate the components

1) Select `cust:panelCustomizable` in the Structure pane.
2) In the Property Inspector, change the **Layout** property to **horizontal**.

This layout change is reflected immediately on the page in the Visual Editor.

3) Scroll down to **Actions**, and set **IsSeededInteractionAvailable** to **true**.

   **Note:** The IsSeededInteractionAvailable attribute controls whether to allow the site administrator to hide and show components.

4) Select **cust:showDetailFrame** in the Structure pane.

5) Update the properties for **cust:showDetailFrame** as described below:
Oracle WebCenter: A Practical Introduction

a) Set **Background** to **dark**. By default, "dark" will be implemented as dark blue.

b) Set **Text** to **Countries to Explore**. The Text attribute defines the title, which is displayed in the header.

6) To give the OmniPortlet a similar look, select it in the Structure pane, and then set the **Background** property to **dark**.

7) To insert white space between the table and portlet, follow these steps:
Oracle WebCenter: A Practical Introduction

a) Return to the ADF Faces Core palette.

b) Click and drag objectSpacer and drop it between the components on the Tours page in the Visual Editor.

Run and test your page

1) Stop the embedded OC4J by clicking the red square in the Embedded OC4J Server log.

2) Run the page (right-click Tours.jspx in the Visual Editor, and then select Run from the context menu) and notice the following:

- The components are laid out horizontally.
- The showDetailFrame has a meaningful title.
- The header color of both components is dark blue.
- White space visually separates the components.
3) Hover the mouse over the top right of the application and notice that the panelCustomizable control appears.

4) Click the down arrow on the control to look at the menu options.
Oracle WebCenter: A Practical Introduction

Note: The control enables the site administrator to customize the page to show or hide content.

Add tabs to the Tours page
To prepare for subsequent practices, you will add three tabs to the Tours page: Tours, Images, and Find Your Tour!

1) On the ADF Faces Core palette, click and drag the ShowOneTab component, and drop it in h:form tab in the Structure pane.

![Structure pane showing ShowOneTab component](image1)

2) Click and drag the ShowDetailItem component from the ADF Faces Core palette, and drop it on the af:showOneTab component in the Structure pane.

![Structure pane showing ShowDetailItem component](image2)

3) Add a two more ShowDetailItem components to af:showOneTab in the Structure pane. You should now have three tabs displayed in the Visual Editor.

```
<table>
<thead>
<tr>
<th>showDetailItem</th>
<th>showDetailItem</th>
<th>showDetailItem</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>showDetailItem</td>
<td>showDetailItem</td>
<td>showDetailItem</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
```

Hands-On Practices: WebCenter p. 36
4) Configure showOneTab to display the tab names only once, above the tab's content:
   a) Select the `showOneTab` component in the Structure pane.
   b) In the Property Inspector, set the `Position` attribute from both to `above`.

   ![Image showing setup steps](image)

   The change takes effect immediately.

5) Rename the tabs:
   a) In the Structure pane, select the first `showDetailItem`.
   b) In the Property Inspector, select the `Text` property, and then change it to `Tours`.
   c) Select the second `showDetailItem`, and then change the `Text` property to `Images`.
   d) Select the third `showDetailItem`, and then change the `Text` property to **Find Your Tour!**.

   ![Image showing renamed tabs](image)
6) In the Structure pane, click and drag `cust:panelCustomizable`, and drop it on the first `showDetailItem`.

![Structure pane with `cust:panelCustomizable` dropped]

At the moment, the other two tabs will be empty.

7) Run the page (right-click Tours.jspx in the Visual Editor, and then select Run from the context menu.), and compare it to the screenshot below.

![Run screenshot showing Viva Venice and Roman Holiday]

*If you have time*

Feel free to work on any of the remaining practices in any order.
Practice 3: Linking the Components

The goal of this practice is to synchronize the tour information displayed in the OmniPortlet (on the right in the figure below) with the country that the user selects in the ADF table (on the left). The country_id, which is a hidden column in the table, will be used to "wire" the two components together.
Look at the predefined page parameters

You will use a page parameter to pass the selected country ID from the table to the portlet. First, let’s open the page definition to see what parameters JDeveloper has already defined.

1) Right-click any empty space on the Tours page, and then select Go to Page Definition from the context menu.

2) In the Structure pane, expand executables; then expand variables.

The variables listed are page variables. When a portlet is added to a page, JDeveloper checks to see if the portlet has public portlet parameters. If it does, JDeveloper automatically adds the corresponding number of page variables and maps them to the portlet parameters.
3) To see OmniPortlet's parameters, expand **OmniPortlet1_1**, and then expand the **parameters** node.

The five parameters that are always predefined for OmniPortlet are listed here.

4) Select the first parameter in the list, **Param1**.

5) Notice that **Param1** is already mapped to the page variable **OmniPortlet1_1_Param1**.
So the portlet parameter will get its value from the page variable OmniPortlet1_1_Param1.

**Set the default value for OmniPortlet1_1_Param1**

To link the tour data displayed in the portlet to the country ID that the user selects, you need to assign the countryId column in the ADF table as the DefaultValue.

1) Click the page variable **OmniPortlet1_1_Param1** in the Structure pane.

![Diagram](image1)

**Tip:** For a better view of **OmniPortlet1_1_Param1** properties, double-click the header of the Property Inspector.

2) Click in the **DefaultValue** field, and click the button to access JDeveloper's Expression Builder.

![Diagram](image2)

3) Expand **ADF Bindings**.

4) Find **CountriesView1** under bindings, and then use the arrow to include it in the binding expression that we want to build.

![Diagram](image3)
5) At the cursor insertion point, add a period (.) followed by the column name `CountryId`.

```
Expression:
${bindings.CountriesView1.CountryId}
```

6) Click OK.

7) Double-click the Property Inspector header to revert it to normal size.

**Customize the portlet to use the parameter**

1) Stop the embedded OC4J by clicking the red square in the Embedded OC4J Server log.

2) Right-click `Tours.jspx` in the navigator, and then select Run.

3) Hover the mouse over the top right of the Veeva Vacations portlet, and click the X on the panelCustomizable control to dismiss it.

4) Click the portlet’s control and select Customize.

5) On the Source tab, set the default value for Param1 to 1001, which is the country ID for France. Click Apply.
6) On the **Filter** tab, update the condition to use `##Param1##` instead of a hard-coded value. Click **Apply**.

![Filter tab with condition](image)

**Note:** Enclosing Param1 with double hash signs (##) forces OmniPortlet to evaluate the parameter.

The picture below shows what's happening—the selected country ID (1010 in the example below) is passed to the page variable as its default value.

![Page variable and portlet parameter](image)

Because the portlet parameter, Param1, is mapped to the same page variable, it receives the value of the country ID. Assuming the user selected India, which has a country ID of 1010, the condition that you just specified ensures that `country_id` equals 1010 is true for the data currently displayed in the portlet.

7) Click **OK** to save your portlet customizations.

**Test that the components are synchronized**

1) Notice that the default value of 1001 is working because tours of France are displayed.
2) Select Indonesia and click Submit.
You should see two Bali tours displayed.

3) Select **India** for a final test.

**Note:** Not all countries have tours.
You may be thinking, "Why do I need to click Submit?" Good question! Let's make a couple of additional changes to fully synchronize the portlet data with the radio button selection.

### Automate the component synchronization

#### Enable AutoSubmit for the radio button

1) In JDeveloper's Visual Editor, click a radio button in the table on the Tours.jspx page.

2) In the Property Inspector, click the AutoSubmit attribute and select true.

#### Set the PartialTrigger attribute of the OmniPortlet

The other change you should make is to the PartialTrigger attribute of the OmniPortlet. PartialTrigger is a component attribute used in partial page rendering (where an area of a page can be refreshed without the need to redraw the entire page). The value of this attribute is the ID (or IDs) of other components. When a component with the matching ID is updated, this component will also be updated.
1) To find the table's ID, click the \texttt{af:table} component in the Structure pane.

![Structure pane with af:table component highlighted]

2) Find the \texttt{Id} in the Property Inspector. You can see that the table ID is \texttt{table1}.

![Property Inspector showing the Id property]

3) Click \texttt{adfp:portlet} in the Structure pane to select it.

![Structure pane with adfp:portlet component highlighted]
4) In the Property Inspector, set the PartialTrigger attribute to **table1**.

![Property Inspector](image.png)

**Note:** The update you just made makes the ADF table a *trigger component* of the portlet. This means, whenever the table is updated, the target component (that is, the portlet) will be updated, too.

**Test your changes**

1) Stop the embedded OC4J by clicking the red square in the Embedded OC4J Server log.

![OC4J Server Log](image.png)

2) Right-click **Tours.jspx** in the Visual Editor, and then select **Run** from the context menu.

3) Select various countries (tour data is available for Australia, France, Italy, Indonesia, India, and New Zealand only) and notice that the tour data in the portlet is refreshed automatically.

**If you have time**

Feel free to work on any of the remaining practices in any order.
Practice 4: Integrating Content from a JCR 1.0 Repository

In this practice, you'll publish file and folder content in a hierarchal tree format using the `getItems` method. Specifically, you will do the following:

- Create a content repository data control.
- Add an ADF Faces tree to display a list of directories and files.
- Use a switcher component for branching.
- Display folders as plain text and file names as hyperlinks.

This is how the Images tab of the Tours page will look after you complete this practice:

When you click the name of a file in the directory structure (which is actually an ADF Faces tree), the relevant picture is displayed. Clicking another file link causes a different picture to be displayed.
Create a file system data control
First, you define a data control that exposes a directory structure.

1) In the navigator, right-click the Model project, and then select New from the context menu.

2) In the New Gallery, select Content Repository. Content Repository Data Control is automatically highlighted. Click OK.

The Create Data Control Wizard opens.

3) On the Welcome page, click Next.

4) On the Data Control Name page, name the data control tourImages.
5) On the Content Repository Configuration page:
   a) Select **File System** for the Repository Type.
   b) Enter `<drive letter>:\Veeva` for the Base Path.

   ![Create Data Control - Step 2 of 3: Content Repository Configuration](image)

   **Note:** The path notation is appropriate for the operating system. In the screen shot above, the Veeva directory is located on the F drive of the PC.

   c) Click **Test** to verify the base path. Then click **OK** to acknowledge the message.

   ![Create Data Control - Test Connection to Content Repository](image)

   d) Click **Finish**.

**Create an ADF Faces tree based on the data control**

You will add the ADF Faces tree to the Images tab, so first you need to make Images the active tab.
**Make the Images tab the active tab**

The Disclosed property of showDetailItem controls which tab is displayed in the Design view and also which tab is shown initially when you run the page.

1) Click **Images** in the Visual Editor. The broken blue border indicates that Images is selected.

2) In the Property Inspector, change the **Disclosed** attribute to **True**.

**Note:** If Disclosed = true for more than one showDetailItem, this condition applies to the first one.

The Images tab should now be the tab displayed in the Visual Editor. If not, select the Tours tab and set the Disclosed property to false.
Create the ADF Faces tree

1) Click the Data Control tab (top right).

2) Expand the tourImages component; then expand the getItems method. This method returns files and folders stored in a specific location of the content repository.

3) Click and drag Return, and drop it on the Images tab on the page.

4) From the Create context menu, select Trees > ADF Tree.
5) In the Action Binding Editor, you need to define the path parameter to point to the directory that is your starting point. Enter a forward slash (/) to indicate that you want to display everything under the base path of the data control.

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

![Action Binding Editor](image)

**Note:** When you defined the base path for your data control, you used Windows notation to specify the path (for example, F:\Veeva). In the Action Binding Editor, you're defining how to use the data control—this information is operating system independent.

6) Click **OK**. The Tree Binding Editor opens.

**Note:** The Tree Binding Editor is used to define rules for a tree data control. A **rule** specifies how the tree data control fetches and displays hierarchical data.

7) In the Tree Binding Editor:
   a) Use the [Ctrl] key to multiselect the **name**, **URI**, and **primaryType** display attributes.

   **Note:** The **URI** (Uniform Resource Identifier) will be used to define hyperlinks to the image files; **primaryType** will be used to differentiate between files and folders in the tree.

   b) Select **Items** under the Branch Rule Accessor.
c) Click **Add New Rule**.

8) Click **OK** to confirm. Then click **OK** again.
9) Move the ADF tree to the Images tab: in the Structure pane, click and drag `af:tree` – `tree1`, and then drop it on `af:showDetailItem` – `Images`.

Run the page

1) Stop the embedded OC4J by clicking the red square in the Embedded OC4J Server log.

2) Run the page to see how the ADF tree looks—right-click `Tours.jspx` in the Visual Editor, and then select Run from the context menu.

3) Click the Images tab.

4) For each top-level folder, notice that the three attributes you selected are displayed: name, URI, and primaryType.
5) Expand **Europe**. Then expand **France** to expose the file names. Notice the primary type of the file names is **nt:file**, not **nt:folder**.

Make the file names hyperlinks

The goal of this practice is to be able to click a file name and have the image in that file appear on the page. To accomplish this, you need to make the file names into hyperlinks that access the relevant images. You will use the URI to link to each image and the primaryType to determine whether a node is a folder or file.

Add a property to the managed bean

1) In the navigator in JDeveloper, in the **ViewController** project, expand **Application Sources** and then **view.backing** to find the managed bean, **Tours.java**.

2) Double-click **Tours.java** to open it.

You will add a property (imageURI) to the managed bean. To do this, you'll add an import statement, define a variable, and add getter and setter methods. You can type the code in directly or copy it from **VeevaSetup\AddToTours.java**.

3) Expand the import node, and then add the following import statement to the bean:

**Hands-On Practices: WebCenter p. 58**
import java.net.URI;
import java.net.URISyntaxException;

import javax.faces.component.html.HtmlForm;
import oracle.adf.view.faces.component.core.data.CoreTree;
import oracle.adf.view.faces.component.core.output.CoreMessages;
import oracle.adf.view.faces.component.core.output.CoreOutputText;
import oracle.adf.view.faces.component.html.HtmlForm;
import oracle.adf.view.faces.component.html.HtmlHead;
import oracle.adf.view.faces.component.html.HtmlBody;
import java.net.URI;
import java.net.URISyntaxException;

4) Scroll down to the variable declarations. Insert the following line just below the last private variable:

private URI imageURI;

5) Add the getter and setter methods for imageURI. You can copy and paste the text below or copy the code from the \VeeveSetup\AddToTours.java file.

// Add these methods:
public void setUri (URI uri) {
    this.imageURI = uri;
}
public URI getUri()
{
    if (this.imageURI == null || this.imageURI.equals ("")) {
        try {
            setUri (new URI ("/veevavacation.jpg");
        }
        catch (URISyntaxException use) {
            System.out.println ("Incorrect default image URI: ");
            use.printStackTrace();
        }
    }
    return this.imageURI;
}
If the user hasn't selected an image yet, `this.imageURI` will be empty, so we'll display a default image, `veevavacation.jpg`. Otherwise, the image that the user selects will be displayed.

**Add veevavacation.jpg to the ViewController project**

1) Open Windows Explorer.

2) Find your JDeveloper home directory, for example, C:\jdev.

3) Navigate to the `<jdev home>\jdev\mywork\veeva\ViewController\public_html` directory.

4) Copy `VeevaSetup\veevavacation.jpg` to the `public_html` directory.

---

Add an objectImage component to the Images tab

---

Hands-On Practices: WebCenter p. 60
Next, you will add an objectImage component to the Tours page. This component will display the image that the user selects.

1) Click the **Tours.jspx** tab.

2) Click the **Components** tab to return to the **ADF Faces Core** palette.

3) On the ADF Faces Core palette, click and drag **ObjectImage**, and then drop it in the form on the **af:showDetailItem – Images** (which is the Images tab) in the Structure pane.

4) When the Insert ObjectImage dialog opens, click **Bind**...
5) In the Bind to Data dialog, find **JSF Managed Beans > backing_Tours > uri**.

6) Click **uri**.

7) Click the arrow to create the EL expression **#{backing_Tours.uri}**.

**Note:** The objectImage component will display the image that is identified by the value of the uri attribute in the managed bean.

8) Click **OK**. Then click **OK** again.
The default image (veevavacation.jpg) should be displayed under the tree.

![Image of a tree with a default image]

**Note:** Initially, you may see a broken link icon – don’t be concerned.

### Add a command link

Next, you will add a commandLink component to represent the file name as a link. The command link submits a request when activated.

1) Click the **Source** tab of the Visual Editor.

2) Find the ADF Faces tree definition in the code. Look for `<af:tree` in the code.

3) Click and drag **CommandLink** from the component palette, and drop it after the `outputText` component of the ADF Faces tree.
4) Change `af:outputText value` to "#{node.URI}".

5) Change `af:commandLink text` to "#{node.name}".

6) Click the Design tab.

7) Double-click a commandLink node to open the Bind Action Property editor. (Alternatively, in the Tours.jspx - Structure window, right click the `af:commandLink - #{node.name}` and select Bind Action Property.)
8) Click **OK** to create a `commandLink` action.

![Bind Action Property](image)

**Note:** Tours.java now appears in the editor, showing the newly created action

```java
public CoreCommandLink getCommandLink1() {
    return commandLink1;
}

public String commandLink1_action() {
    // Add event code here...
    return null;
}
```

9) Click the **Tours.jspx** tab to return to the Visual Editor.

**Add a switcher component**

Next, you'll add an ADF switcher component to the tree. This component allows the application to render the tree elements differently, depending on whether the node type is folder or file.

1) In the Structure pane, scroll down to the ADF Faces tree.

![Tours.jspx - Structure](image)
2) Right-click the tree's **nodeStamp** facet, and then select **Insert inside nodeStamp** > **ADF Faces Core** from the context menu.

![Diagram of Oracle WebCenter Structure]

**Note:** A JSF **facet** tag is used to hold other components that require a special relationship with the parent component. The **nodeStamp** facet of a tree is used to display the data for each element in the tree. The tree does not create components for each element; instead, the nodeStamp is repeatedly rendered (or "stamped") once per element.

3) In the Insert ADF Faces Core Item dialog, select **Switcher** and click **OK**.

![Insert ADF Faces Core Item dialog]

**Add two facets to the switcher**

Next, you'll add two facets to the switcher component: one for folders, the other for files.

1) Right-click the switcher component, and then select **Insert inside af:switcher** > **JSF Core** > **Facet** from the context menu.

![Diagram of Insert ADF Faces Core Item dialog]
2) Name the first facet `nt:folder`, and then click **OK**.

![Insert Facet](image1)

3) To add a second facet, right-click the switcher component, and then select **Insert inside af:switcher > JSF Core > Facets** from the context menu.

4) Name the second facet `nt:file`, and then click **OK**.

![Insert Facet](image2)

5) Right-click the switcher component, and then select **Properties**.

6) Enter `#{node.primaryType}` for the **FacetName**. This EL expression, when evaluated at runtime, determines the facet to be rendered. In this case, the primaryType of the node will determine how the node is rendered.

![Switcher Properties](image3)

7) On the Advanced Properties tab, enter `nt:file` for the **DefaultFacet** value, and then click **OK**.

![Switcher Properties](image4)

Create an outputText component

To display folder names in the tree, you'll create a new outputText component, which will be inserted inside the nt:folder facet.

1) From the context menu for the `nt:folder` facet, select **Insert inside f:facet - nt:folder > ADF Faces Core**.
2) In the Insert ADF Faces Core Item, select **OutputText**, and then click **OK**.

![Insert ADF Faces Core Item](image1)

3) In the Property Inspector, modify the **Value** property of the outputText component to **#{node.name}**.

![Output Text - outputText1 - Property Inspector](image2)

4) In the Structure pane, click and drag the **commandLink**, and drop it on the **nt:file** facet.

![Tours.jpx - Structure](image3)

Next, you will define a SetActionListener for the commandLink. The SetActionListener component allows you to set parameters.
5) Right-click `af:commandLink`, and then select `Insert inside af:commandLink - #{node.name} > ADF Faces Core` from the context menu.

6) In the Insert ADF Faces Core Item dialog, select `SetActionListener`, and then click `OK`.

7) Define the common properties for the SetActionListener:
   a) In the `From` field, enter `#{node.URI}`.
   b) In the `To` field, enter `#{backing_Tours.uri}`.
   c) Click `OK`.

8) Move `af:outputText - #{node.URI}` to the switcher component.
Check your work

1) Compare the components in the Images tab of your Structure pane with the solution below. If necessary, make any corrections before you run the page.

![](image)

**Tip:** For improved visibility, double-click the Tours.jspx – Structure tab to expand it to full screen.

2) To better understand what you’ve done in this practice, read the information below, which summarizes the purpose of each component in the ADF Faces tree.

<table>
<thead>
<tr>
<th>Component</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>af:switcher</code></td>
<td>Provides branching on primaryType: file or folder</td>
</tr>
<tr>
<td><code>f:facet - nt:folder</code></td>
<td>Holds the components for handling folders</td>
</tr>
<tr>
<td><code>af:outputText - #{node.name}</code></td>
<td>Displays the folder name</td>
</tr>
<tr>
<td><code>f:facet - nt:file</code></td>
<td>Holds the components for handling files</td>
</tr>
<tr>
<td><code>af:commandLink - #{node.name}</code></td>
<td>Displays the file name as a hyperlink</td>
</tr>
<tr>
<td><code>af:setActionListener</code></td>
<td>Gets the value of the URI of the currently selected item and sets it as the value for the uri attribute in the managed bean</td>
</tr>
<tr>
<td><code>af:outputText - #{node.URI}</code></td>
<td>Displays the image identified by the value of the uri attribute in the managed bean</td>
</tr>
</tbody>
</table>

Run the page

1) Stop the embedded OC4J by clicking the red square in the Embedded OC4J Server log.

![](image)

2) Run the page to see the impact of your changes—right-click Tours.jspx in the Visual Editor, and then select Run from the context menu.
3) At the top level, notice the folder names now appear as simple names, not hyperlinks.

4) Expand **Europe**; then expand **France**.

5) Click any of the tours in the **France** directory.
Notice the relevant image that is displayed.

6) (Optional) Look at the pictures for tours in Italy, India, and Indonesia.  
   **Note:** The countries that are not listed here have no tour information.

**If you have time**

Feel free to work on any of the remaining practices in any order.
Practice 5: Building and Deploying Standards-Based Portlets

In this practice, you create a standards-based portlet to display a list of tours. Then, after adding the Parameter Form portlet, which is available preinstalled in the WebCenter Preconfigured OC4J, you will wire the two portlets together by means of a page parameter.

The screen shot below shows the two portlets. After the user enters an airport code and clicks OK, the list of tours is displayed in the Scheduled Tours portlet.
Create the ListTours portlet

**Create a simple portlet using a wizard**

1) Right-click the **Portlets** project, and then select **New**.

2) In the New Gallery, select **Portlets**, and then select **Standards-based Java Portlet (JSR 168)**.

3) Click **OK**. The JSR 168 Java Portlet wizard opens.

4) On the Welcome page, click **Next**.
5) In Web Application page, click **Next** to create a J2EE 1.4 application.

6) On the General Portlet Properties page, do the following:
   a) Name the portlet **ListTours**.
   b) Name the class **ListTours**.
   c) Enable WSRP v2 extensions.

**Note:** Oracle WebCenter supports an early version of WSRP 2.0. JSR 168 does not support interportlet communication, but, by enabling the WSRP 2.0 extensions, you
allow the container to work around this limitation of the Java API by handling interportlet communication.

7) Click Next.

8) On the Name and Attributes page, keep the default name and titles. Add a description such as “Lists tours available from the selected airport.” Click Next.

9) On the Content Types and Portlet Modes page, click view to see how the view mode will be implemented. Click Next.
10) In the Portlet Title field, assign a default value of **Scheduled Tours**.

11) Click **Next** four times to navigate to the **Portlet Navigation Parameters** page.

12) On the Portlet Navigation Parameters page, add a parameter named airport:
   
a) Click **Add**.
   
b) Enter **airport** for the parameter name.
   
c) Enter **airport** for the parameter label.

   **Tip:** If you do not see the Portlet Navigation Parameters page, you need to return to page 2 of the wizard (General Portlet Properties) and select WSRP v2 extensions.

13) Click **Next**, and then click **Finish**.

**Look at the portlet**

1) In the Applications Navigator, expand Portlets > Application Sources. Then expand portlet.ListTours.java is the Java class that serves as the controller file for the portlet.
2) Expand **portlet.resource**. **ListToursBundle.resource** is the portlet's resource bundle, which facilitates localization.

3) Expand **Web Content** and **ListTours.html**. The two files listed (*edit.jsp* and *view.jsp*) are implementations of the two portlet modes that your portlet supports.

4) Double-click **view.jsp** to open it in the Visual Editor. As you can see, the portlet will display a simple message to the user, "Welcome, this is the view mode."
Before you make any changes, let's deploy the portlet.

**Deploy the portlet**
The first step is to define a connection to an application server or standalone OC4J. You can skip the next 8 steps if you have already defined the LocalOC4J connection in another practice.

**Define an application server connection**
Before you define the application server connection, be sure that your preconfigured OC4J is running.

1) Click the **Connections** tab in the navigator.
2) Right-click **Application Server**, and then select **New Application Server Connection**.
3) On the Welcome page, click **Next**.
4) Name the new connection **LocalOC4J** and click **Next**.
5) Let the **Username** default to `oc4jadmin` and enter `welcome` for the **Password**. Click **Next**.

![Create Application Server Connection - Step 2 of 4: Authentication](image)

### Create Application Server Connection - Step 2 of 4: Authentication

Specify a Username and password to authenticate the connection. To bypass authentication at runtime, select Deploy Password.

- **Username:** \( oc4jadmin \)
- **Password:** \( \text{********} \)

Click **Next**.

6) Accept the default Host Name of `localhost`, but modify the **RMI Port** to **22667**, which is the default port for the Preconfigured OC4J.

![Create Application Server Connection - Step 3 of 4: Connection](image)

### Create Application Server Connection - Step 3 of 4: Connection

Please provide the host name and RMI port for the OC4J 10.1.3 instance. The information is used to assemble an URL used to create a JMX connection to the server. An additional path may be specified as part of the URL if required; by default it is not necessary.

- **Host Name:** \( localhost \)
- **RMI Port:** 22667

Click **Next**.

7) Click **Test Connection** and wait for the **Success!** message.

![Create Application Server Connection - Step 4 of 4: Test](image)

### Create Application Server Connection - Step 4 of 4: Test

Click Test Connection to determine if the information specified successfully establishes a connection with the application server.

- **Status:** Success

8) Click **Finish**.

**Create a deployment profile**

Next, you define a deployment profile, which is a JDeveloper project component that governs the deployment of a J2EE Web module to an application server.

1) Click the **Applications Navigator** tab (top left).
2) Right-click **Portlets**, and then select **New**.
3) Select Deployment Profiles and WAR File.

Note: A WAR file (Web archive) encapsulates in a single file all the components that are necessary to run an application.

4) Name the deployment profile **ListTours** and accept the default directory. Click **OK**.
5) Click Specify J2EE Web Context Root and enter ListTours.

![War Deployment Profile Properties](image1)

**Note:** The context root is the entry that follows the port number in a URL that accesses the portlet. For example, http://<host>:<port>/<context root>/

6) Click OK.

7) A new node (Resources) has been added to the Portlets project. Expand it to see the deployment profile.

![Applications Navigator](image2)

**Deploy the portlet**

You’re ready to deploy the portlet. Actually, you deploy the portlet producer. A producer can contain one or more portlets, and provides the communication link between the portlet consumer (that is, a WebCenter application or a portal) and the portlets it consumes.

1) Right-click the deployment profile and select **Deploy to > LocalOC4J**, which is the connection you just defined.

2) In the Configure Application dialog, click **OK**.
3) Look for the “Deployment finished” message in the deployment log.

![Deployment-log](image)

**Test the deployment**

Before you register a portlet, it’s a good idea to test that the portlet has been successfully deployed and that you can access the WSDL.

1) Open a browser.

2) Go to the WSRP Producer Test Page. The format for the URL is: http://<host>:<port>/<context root>/info. For example, http://localhost:6888/ListTours/info.

3) On the test page, click **WSRP v2 WSDL**.
4) The portlet you just deployed has been exposed as a Web service. In your browser, notice the Web Services Description Language (WSDL) that describes this Web service.

```
<wsdl:definitions targetNamespace="urn:oasis:names:tc:wsrp:v2:wsdl">
  <import namespace="urn:oasis:names:tc:wsrp:v2:bind" location="wsrp_v2_bindings.wsdl"/>
  <wsdl:service name="WSRP_v2_Service">
    <wsdl:port binding="bind:WSRP_v2_ServiceDescription_Binding_SOAP" name="WSRP_v2_ServiceDescription_Service">
      <soap:address location="http://localhost:6688/ListTours/portlets/WSRP_v2_ServiceDescription_Service"/>
    </wsdl:port>
    <wsdl:port binding="bind:WSRP_v2_Markup_Binding_SOAP" name="WSRP_v2_Markup_Service">
      <soap:address location="http://localhost:6688/ListTours/portlets/WSRP_v2_Markup_Service"/>
    </wsdl:port>
    <wsdl:port binding="bind:WSRP_v2_Registration_Binding_SOAP" name="WSRP_v2_Registration_Service">
      <soap:address location="http://localhost:6688/ListTours/portlets/WSRP_v2_Registration_Service"/>
    </wsdl:port>
    <wsdl:port binding="bind:WSRP_v2_PortletManagement_Binding_SOAP" name="WSRP_v2_PortletManagement_Service">
      <soap:address location="http://localhost:6688/ListTours/portlets/WSRP_v2_PortletManagement_Service"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

5) Copy the URL to the clipboard. You will use it to register the portlet producer.

```
http://localhost:6688/ListTours/portlets/wsrp2?WSDL
```
Consume the portlet

Register the portlet producer

1) Right-click ViewController in the navigator, and then select New from the shortcut menu.

Note: Registration is part of consuming a portlet, so you should register the portlet in the ViewController project, not the Portlets project. The Portlets project is for producing portlets—that is, creating, enhancing, and deploying them—whereas the ViewController project is used to create the JavaServer Faces pages for the UI of your application and to consume portlets.

2) Click Portlets, and then click WSRP Producer Registration.

3) Click OK to launch a wizard to register your WSRP producer.

4) On the Welcome page, click Next.
5) On the Name page, rename the producer **VeevaProducer**, and then click **Next**.

6) On the **Connection** page, paste the URL from your browser as the **URL Endpoint**. Click **Next**.

The wizard connects to the WSRP portlet producer, using the URL you just entered. **Note:** Your OC4J must be running for this step to succeed.

7) On the **Registration Details** page, accept the default timeout of 30 seconds, and click **Finish**.
8) Click **OK** to acknowledge the successful producer registration.

![Image of successful producer registration dialog]

**Make the Find Your Tour! tab current**

You will add the ListTours portlet to the “Find Your Tour” tab on the Tours page. First you need to make that tab the current tab in the Visual Editor.

1) Ensure that **Tours.jspx** is the currently displayed page in the Visual Editor.

2) In the Structure pane of the Veea application, select **af:showDetailItem – Find Your Tour!**, which is the Find Your Tour! tab.

3) In the Property Inspector, set the **Disclosed** property to **true**.

![Image of Property Inspector with Disclosed property set to true]

4) Set the **Disclosed** property to **false** for the other two tabs. Find Your Tour! is now the active tab.

![Image showing the active Find Your Tour! tab]

**Note:** The Disclosed property of showDetailItem controls which tab is displayed in the Design view and also which tab is shown initially when you run the page. If **Disclosed = true** for more than one showDetailItem, this condition applies to the first one.
Add the portlet to a page

Now that the application knows about the portlet, you can add the portlet to the Tours page.

1) Find the new VeevaProducer in the list of component palettes, and then select it.

2) Click and drag ListTours from the VeevaProducer palette, and drop it on the third showDetailItem in the Structure pane.

You can now see the ListTours portlet on Tours page.
3) Run the page to see how the portlet looks.

Currently, the portlet isn't very exciting, but at least you know it works.

**Add the Parameter Form portlet**
Next, you add a parameter form to the page.

**Add the Parameter Form portlet**
The WebCenter Preconfigured OC4J has a preinstalled WSRP parameter form, which is pictured below. To use it, you simply need to register the producer with your application, and then add the portlet to the page.

1) In the Veeva application in the Applications Navigator, right-click Portlet Producers, and then choose New WSRP Producer from the context menu.

2) On the Welcome page of the registration wizard, click Next.
3) On the Name page, name the producer **WsrpSamplesProducer**. Then click **Next**.

![Register WSRP Portlet Producer - Step 1 of 5: Name](image)

4) On the Connection page, enter the following URL as the Endpoint: `http://localhost:6688/portletapp/portlets/wsrp2?WSDL`. Notice that URL is similar to the one we used to register ListTours; however, the context root is `portletapp`. Then click **Next**.

![Register WSRP Portlet Producer - Step 2 of 5: Connection](image)

5) On the Registration Details page, click **Finish**.

6) Click **OK** to acknowledge the successful registration message.

![Register WSRP Portlet Producer](image)

**Add the portlet to the Tours page**

1) Find the new **WsrpSamplesProducer** in the list of component palettes.

![Component list](image)
2) Select **WsrpSamplesProducer** to see the portlets for the producer. Several portlets are included in WSRP Samples. Scroll down to the **Parameter Form Portlet**.

![Diagram of WsrpSamplesProducer](image1)

3) Click and drag the **Parameter Form Portlet**, and drop it above the **ListTours** portlet in the Structure pane.

**Note:** Both portlets are represented by the same type of tag (**adfp:portlet**).

![Diagram of portlets](image2)

The Parameter Form portlet immediately appears above ListTours in the Visual Editor. To make the page look tidier, you can add some white space between the two portlets, as directed in the next step.

4) Add white space between the portlets:

a) From the Components list, select the **ADF Faces Core** palette.

**Hands-On Practices: WebCenter p. 91**
b) Scroll down to **ObjectSpacer**.

![](image1)

- Click and drag **ObjectSpacer**, and drop it between the two portlets.

Your page should look like this:

![](image2)

**Customize the Parameter Form portlet**

1) Stop the embedded OC4J by clicking the red square in the Embedded OC4J Server log.
2) Run the page so that you can customize the Parameter Form portlet.

3) Notice that the WSRP sample parameter form has three parameters. You need only one parameter—for the airport code. You will customize the portlet to change the title, modify the prompt, and hide the extra parameters.

4) Click the control in the Parameter Form portlet’s header, and then select **Customize** from the menu.

5) Make the following changes to the portlet:
   a) Change the title to **Enter your departure airport**.
   b) Modify the prompt for the first parameter to **Airport Code**.
   c) Remove the prompts of the second and third parameters to hide them.
6) Click OK to save the customizations.

7) Your parameter form should look like this:

![Parameter Form](image)

**Add logic to the ListTours portlet**

Next, you will add some logic to the ListTours portlet, and then link the portlets by means of parameters to synchronize the tour schedule with the chosen airport.

**Create the Tours class**

The first class that you will create is Tours.java, which will contain the getter and setter methods.

1) Right-click the Portlets project, and then select New from the context menu.

![New Gallery](image)

2) In the New Gallery, select General > Java Class, and then click OK.
Name the class Tour and click OK.

3) Double-click the Tour.java tab to maximize it.

4) Replace the current contents of Tour.java with the code in VeevaSetup\ListToursPortlet\Tour.java.
5) Double-click the `Tour.java` tab again to revert the file to its original size.

```
package portlet;
```

**Create the TourData class**

TourData.java is the JavaBean that will define the data for the portlet.

**Note:** You’re using a class rather than database tables for simplicity. Naturally, in a production application, you would retrieve scheduled tour data from your Oracle database.

1) To add a second class to the portlet, right-click the **Portlets** project in the navigator, and then select **New**.
2) In the New Gallery, select Java Class and click OK.
3) Name the class TourData and click OK.

4) Maximize TourData.java.
5) Replace all the code with the contents of VeevaSetup\ListToursPortlet\TourData.java.

```java
new Tour(2015,"Tuscany Tasting",createDate(2007,1,10),10,"SFU"),
new Tour(2017,"Tuscany Tasting",createDate(2007,3,1),10,"SFU"),
new Tour(2019,"Tuscany Tasting",createDate(2007,4,1),10,"SFU")
;

private static Date createDate(int year, int month, int day) {
    return new GregorianCalendar(year, month - 1, day).getTime();
}

public Tour[] getToursFrom(String airport) {
    Vector v = new Vector();
    for (int i = 0; i < allTours.length; i++) {
        if (airport.equals(allTours[i].getAirport())) {
            v.addElement(allTours[i]);
        }
    }
    Tour[] result = new Tour[v.size()];
    for (int i = 0; i < v.size(); i++) {
        result[i] = (Tour) v.elementAt(i);
    }
    return result;
}

public static void main(String[] args) {
    System.out.println(createDate(1916,7,9));
    System.out.println(createDate(2004,12,31));
}
```
6) Scroll up and look at the data. Notice that, in our data, all tours leave from SFO (San Francisco International Airport).

7) Double-click the tab to revert **TourData.java** to its original size.

**Update view.jsp**

You have added two Java classes to the ListTours portlet:

- **Tour**: Creates a Tour object, which contains information about a particular tour; Tour also contains getter and setter methods.
- **TourData**: Builds an array of scheduled tours.

Next, you will update **view.jsp** to display the tour schedule, based on the value of the airport parameter, which you defined when you created the ListTours portlet.

1) If necessary, scroll the tabs horizontally to find **view.jsp**.
2) Maximize `view.jsp`. If the design view is displayed, click the **Source** tab.

3) Use copy and paste to replace the generated code with the contents of `VeevaSetup\ListToursPortlet\view.jsp`. 
4) Scroll up and look at the code. Notice where the portlet gets the **airport** parameter and then uses it to get the tour data for that airport.

```xml
<portlet:defineObjects/>
<string name="data" class="portlet.TourData" scope="request" />

String airport = renderRequest.getParameter("airport");
if ((airport == null) || (airport.length() == 0)) {
    <p class="portlet-text">Please specify an airport code!</p>
}
else {
    Tours[] tours = data.getToursFrom(airport);
    if (tours.length == 0) {
    }
}
```

Then the code iterates through the tours for the specified airport.

```xml
<table>
<tr class="portlet-section-header">
<th colspan="2">Tours available from airport: &lt;airport&gt;</th>
</tr>
<tr class="portlet-section-header">
<th>ID</th><th>Tour Name</th><th>Starting Date</th><th>Duration</th>
</tr>

for (int i = 0; i < tours.length; i++) {
    DateFormat df =
        DateFormat.getInstance(DateFormat.FULL, renderRequest.getLocale());

    <tr class="portlet-section-body">
    <td>&lt;= tours[i].getId()</td>
    <td>&lt;= tours[i].getName()</td>
    <td>&lt;= df.format(tours[i].getDeparture())</td>
    <td>&lt;= tours[i].getDuration()</td>
    </tr>
    }

</table>
```

**Redeploy the Portlet**

Because you've modified the ListTours portlet, you need to redeploy it.

1) Right-click the deployment profile **ListTours.deploy**, and then select **Deploy to LocalOC4J** (which is already preselected).

2) When the Configure Application dialog opens, click **OK**.
3) JDeveloper lets you know that the current version of the portlet producer must be undeployed. Click Yes to confirm.


**Link the portlets together**
The portlet logic is now in place. Next, you need to link the two portlets together by mapping their parameters to the same page variable.

1) Display the Tours.jspx page in the Visual Editor. You may need to scroll the tabs to find it.
2) Right-click the page, and then select **Go to Page Definition**.

3) Find the variable definitions in the XML. JDeveloper has automatically created a page variable for each portlet parameter. The **ListTours1_1_airport** page variable corresponds to the **airport** parameter in the ListTours portlet.
4) The graphic below shows what we want to accomplish: map the portlet parameter coming from the parameter form to the *same page variable* that the airport parameter in the ListTours portlet is mapped to.

5) To find the Parameter Form portlet's parameters in the Structure pane, expand **ParameterFormPortlet1_1**, and then expand **parameters**.
6) Select the first parameter, `ora_wsrp_navigparam_Parameter1`. In the Properties Inspector, you can see that this parameter is mapped by default to its corresponding page variable.

7) Use the drop-down list for the `pageVariable` property to remap the parameter to a different page variable. Select `ListTours1_1_airport`, which is same page variable that the airport parameter is mapped to.

---

**Run and test your portlets**

1) Stop the embedded OC4J by clicking the red square in the Embedded OC4J Server log.

2) Run the page once more to test the interportlet communication.
Enter **SFO** (in upper case) as the airport code, and then click **OK**. You should see the tours departing from San Francisco International Airport listed in the Scheduled Tours (ListTours) portlet.

<table>
<thead>
<tr>
<th>ID</th>
<th>Tour Name</th>
<th>Starting Date</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>1003</td>
<td>Bali Homestay</td>
<td>Monday, January 1, 2007</td>
<td>7</td>
</tr>
<tr>
<td>1004</td>
<td>Bali Culture</td>
<td>Thursday, February 1, 2007</td>
<td>6</td>
</tr>
<tr>
<td>1005</td>
<td>Bali Homestay</td>
<td>Thursday, March 1, 2007</td>
<td>7</td>
</tr>
<tr>
<td>1006</td>
<td>Bali Culture</td>
<td>Sunday, April 1, 2007</td>
<td>5</td>
</tr>
<tr>
<td>1007</td>
<td>Golden Triangle</td>
<td>Monday, January 1, 2007</td>
<td>6</td>
</tr>
<tr>
<td>1008</td>
<td>Golden Triangle</td>
<td>Thursday, February 1, 2007</td>
<td>6</td>
</tr>
<tr>
<td>1010</td>
<td>Darjeeling Delight</td>
<td>Thursday, February 15, 2007</td>
<td>5</td>
</tr>
<tr>
<td>1011</td>
<td>Golden Triangle</td>
<td>Thursday, March 1, 2007</td>
<td>6</td>
</tr>
<tr>
<td>1012</td>
<td>Darjeeling Delight</td>
<td>Thursday, March 15, 2007</td>
<td>5</td>
</tr>
<tr>
<td>1013</td>
<td>Golden Triangle</td>
<td>Sunday, April 1, 2007</td>
<td>6</td>
</tr>
<tr>
<td>1014</td>
<td>Darjeeling Delight</td>
<td>Sunday, April 15, 2007</td>
<td>5</td>
</tr>
<tr>
<td>1015</td>
<td>Tuscany Trekking</td>
<td>Monday, January 1, 2007</td>
<td>10</td>
</tr>
<tr>
<td>1016</td>
<td>Mediterranean Sand &amp; Sea</td>
<td>Thursday, February 15, 2007</td>
<td>5</td>
</tr>
<tr>
<td>1017</td>
<td>Tuscany Trekking</td>
<td>Thursday, March 1, 2007</td>
<td>10</td>
</tr>
<tr>
<td>1018</td>
<td>Mediterranean Sand &amp; Sea</td>
<td>Sunday, April 15, 2007</td>
<td>5</td>
</tr>
<tr>
<td>1019</td>
<td>Tuscany Trekking</td>
<td>Monday, January 1, 2007</td>
<td>10</td>
</tr>
</tbody>
</table>

*If you have time*
Feel free to work on any of the remaining practices in any order.
Practice 6: Portletizing an ADF Faces Application

In this practice, you do the following:

- Create an application that contains a simple page to portletize.
- “Portletize” the page.
- Add a test page to the Veeva application.
- Register and consume the portlet.

The screen shot below shows the simple ADF Faces application, consisting of an ADF Faces table, on the left and the portletized version of the table on the right.

Note: Although the page that you build in this practice is very simple, you can use the Portlet Faces Bridge to portletize significantly more complex pages.
Create a page to portletize
You will create a new JSF application for the portlet producer.

Create an application

1) In the Applications Navigator, right-click Applications, and then select New Application from the context menu.

2) In the Create Application dialog, enter Countries for the Application Name and select Web Application [JSF, ADF BC] as the Application Template. Click OK.

3) In the navigator, notice that two projects that have been created for you, Model and ViewController.

Create Countries.jspx

1) Right-click the View-Controller project, and then select New from the context menu.
2) In the New Gallery, select **JSF**, and then select **JSF JSP**. Click **OK**.

![New Gallery](image1.png)

3) On the Welcome page, click **Next**.

4) Name the file **Countries.jspx**, and then select **JSP Document**. Click **Finish**.

![Create JSF JSP - Step 1 of 4: JSP File](image2.png)

*Add the Countries table to the page*

**Countries.jspx** should already be open in the Visual Editor.
1) Click the **Connections** tab (top left).

2) Under the Database node, find **Vacations > VEEVA > Tables > COUNTRIES**.

3) When the Create context menu opens, select **Tables > ADF Read-only Table**.
4) In the **Edit Columns** dialog, keep all the columns and enable selection and sorting. Click **OK**.

![Edit Table Columns dialog](image1)

**Restrict the number of rows displayed**
Because you will portletize the page, you should restrict the number of rows displayed to keep the portlet content small.

1) Click the **Go to Page Definition** link (below the Property Inspector). **Tip**: If you don’t see the Property Inspector, click anywhere on the page in the Visual Editor.

![Table - table1 - Property Inspector](image2)

2) Change the **RangeSize** from 10 to **5**.

3) Run the page.
Your table should look like the one below.

<table>
<thead>
<tr>
<th>Select</th>
<th>and</th>
<th>Submit</th>
<th>Previous</th>
<th>1-5 of 12</th>
<th>Next 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>CountryID</td>
<td>CountryName</td>
<td>Continent</td>
<td>Currency</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1001</td>
<td>France</td>
<td>Europe</td>
<td>Euro</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1002</td>
<td>Spain</td>
<td>Europe</td>
<td>Euro</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1003</td>
<td>USA</td>
<td>North America</td>
<td>US dollar</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1004</td>
<td>Hungary</td>
<td>Europe</td>
<td>Hungarian forint</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1005</td>
<td>Indonesia</td>
<td>Asia</td>
<td>Indonesian rupiah</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Portletize the page**

Portletizing an ADF Faces application requires just two additions to the application:

- A library
- A portlet deployment descriptor

**Add the JSF portlet bridge to the project**

1) In the Applications Navigator, double-click the ViewController project to open the Project Properties dialog.

2) Select Libraries, and then click Add Library.

3) Scroll down to Portlet Faces Bridge.
4) Select Portlet Faces Bridge and click OK.

5) In the Project Properties dialog, click OK.

Create a portlet deployment descriptor
Next, you will create a portlet deployment descriptor that enables you deploy the page as a portlet producer.

1) Right-click ViewController, and then select New from the context menu.

2) In the New Gallery, change Filter By to All Technologies so that you can see more categories.

3) Expand the General node.

4) Under the General category, select Deployment Descriptors, and then select portlet.xml (Portlet Deployment Descriptor).

5) Click OK.
Replace the code in portlet.xml

1) In the navigator, expand ViewController > WebContent >WEB-INF. Double-click portlet.xml to open it.

2) You will replace the default code with a code sample provided by the Oracle WebCenter Framework Developer's Guide. For your convenience, we’ve included Example 17-13 in the Developer’s Guide in the VeevaSetup\portlet.xml file.

3) In JDeveloper, highlight and cut the default code in portlet.xml.

4) Copy the contents of VeevaSetup\portlet.xml to the clipboard.

5) Paste the code in portlet.xml in JDeveloper.
Your `portlet.xml` should look like the example below.

```
<portlet version="1.0" encoding="UTF-8">

  <portlet-app version="1.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd">
    <portlet>
      <description>ADF Faces Demo Portlet</description>
      <portlet-name>ADFFacesDemo</portlet-name>
      <display-name>ADF Faces Demo portlet</display-name>
      <portlet-class>oracle.portlet.server.bridges.jsf.FacesPortlet</portlet-class>
      <init-param>
        <name>DefaultPage.view</name>
        <value>/index.jsp</value>
      </init-param>
      <init-param>
        <name>BridgeLifecycleListeners</name>
        <value>oracle.portlet.server.bridges.jsf.ADFacesBridgeLifecycleListener, oracle.portlet.server.bridges.jsf.ADFacesBridgeLifecycleListener</value>
      </init-param>
      <supports>
        <name-type>text/html</name-type>
        <portlet-mode>VIEW</portlet-mode>
      </supports>
      <portlet-info>
        <title>ADF Faces Demo Portlet</title>
        <short-title>ADFFacesDemo</short-title>
        <portlet-info>
          <portlet>
            <init-param>
              <name>DefaultPage.view</name>
              <value>/Countries.jsp</value>
            </init-param>
          </portlet>
        </portlet-info>
      </portlet-info>
    </portlet>
  </portlet-app>
```

Modify `portlet.xml`

You will modify the values of various tags to suit the Countries page.

1) Update the basic portlet information:

```
<description>Veeva Countries portlet</description>
<portlet-name>VeevaCountries</portlet-name>
<display-name>Veeva Countries portlet</display-name>
```

2) Update the `DefaultPage.view` parameter, which specifies the default page for the application to use as the portlet's view mode. Its location is relative to the web-app-context-root and always starts with a `/`. Be sure the name in the `<value>` tag matches the name of your JSP.

```
<init-param>
  <name>DefaultPage.view</name>
  <value>/Countries.jsp</value>
</init-param>
```
3) Update the title tag values:

```xml
<portlet-info>
  <title>Veeva Countries Portlet</title>
  <short-title>Countries</short-title>
</portlet-info>
```

**Deploy the portlet**

*Create a deployment profile*

1) Right-click the ViewController project in the navigator, and then select New.

2) In the New Gallery, select General > Deployment Profiles and WAR file.

3) Click OK.

4) In the Create Deployment Profile – War File dialog, name the profile countries and click OK.
5) Select **Specify J2EE Web Context Root** and enter **countries**.

![Image of WAR Deployment Profile Properties]

6) Click **OK**.

**Deploy the portlet producer**

1) In the navigator, expand the **Resources** node in the **ViewController** project to access the deployment profile, **countries.deploy**.

2) Right-click **countries.deploy**, and then select **Deploy to > LocalOC4J**. LocalOC4J is an application server connection to the Preconfigured OC4J, which must be running during the deployment.

![Image of Deployment Options]

3) When the Configure Application dialog opens, click **OK**.

4) Wait for the **Deployment finished** message in the deployment log.

![Deployment Log]

**Hands-On Practices: WebCenter p. 116**
5) Now you can visit the portlet producer's test page. Open a browser, and then enter the following URL:

   http://localhost:6688/countries/info

6) Click *WSRP v2 WSDL*.

   **Note:** If you were planning to consume the portlet in Oracle Portal, you should click *WSRP v1 WSDL*. WSRP v2 is an early implementation of the standard, which is supported in WebCenter applications exclusively.

   ```xml
   <definitions targetNamespace="urn:oasis:names:tc:wsrp:v2:wsdl">
     <import namespace="urn:oasis:names:tc:wsrp:v2:bind" location="wsrp_v2_bindings.wsdl"/>
     <service name="WSRP_v2_Service">
       <port binding="bind:WSRP_v2_ServiceDescription_Binding_SOAP" name="WSRP_v2_ServiceDescription_Service">
         <soap:address location="http://localhost:6688/countries/portlets/WSRP_v2_ServiceDescription_Service"/>
       </port>
       <port binding="bind:WSRP_v2_Markup_Binding_SOAP" name="WSRP_v2_Markup_Service">
         <soap:address location="http://localhost:6688/countries/portlets/WSRP_v2_Markup_Service"/>
       </port>
       <port binding="bind:WSRP_v2_Registration_Binding_SOAP" name="WSRP_v2_Registration_Service">
         <soap:address location="http://localhost:6688/countries/portlets/WSRP_v2_Registration_Service"/>
       </port>
       <port binding="bind:WSRP_v2_PortletManagement_Binding_SOAP" name="WSRP_v2_PortletManagement_Service">
         <soap:address location="http://localhost:6688/countries/portlets/WSRP_v2_PortletManagement_Service"/>
       </port>
     </service>
   </definitions>
   ``

7) Copy the URL to the clipboard to use for registering the portlet producer.

   ![http://localhost:6688/countries/portlets/WSRP2?WSDL]
Consume the portlet
You will consume the application in a new test page in the Veeva application.

Register the portlet
1) In the Veeva application, right-click the ViewController project, and then select New from the context menu.
2) In the New Gallery, select Web Tier > Portlets and WSRP Producer Registration.
3) Click OK to launch the registration wizard.
4) On the Welcome page, click Next.
5) On the Name page, enter CountriesProducer as the name, and then click Next.
6) In the Connection page, paste the URL you copied from the browser into the URL Endpoint field, and then click Next.

The wizard creates a connection to the producer.

7) On the Registration Details page, click Finish.
8) Click **OK** to acknowledge the successful registration.

---

**Add the portlet to a test page**

1) In the Veeva application in the Applications Navigator, right-click **ViewController**, and select **New** from the context menu.

2) In the New Gallery, select **Web Tier > JSF**, and then select **JSF JSP** in the Items pane.

3) Click **OK**. The Create JSF JSP Wizard opens.
4) On the Welcome page, click **Next**.

5) Name the file **TestPage.jspx**, and then select **JSP Document**, which is an XML representation of a JSP page. Click **Finish**.

![Create JSP JSP - Step 1 of 4: JSP File](image)

**Add the portlet to the page**

1) In the list of component palettes, find **CountriesProducer** and select it.

2) Click and drag **Veeva Countries Portlet** and drop it on **TestPage.jspx**.

![Components Palette](image)
3) Stop the embedded OC4J by clicking the red square in the Embedded OC4J Server log.

4) Run the page.

Your portlet should look like this one:

*If you have time*
Feel free to go back and work on any practices that you may have skipped.