Creating and Using Templates for the Content Presenter Task Flow

Integrating content stored in Oracle WebCenter Content Server is one of the most often used services in an Oracle WebCenter Portal application. Similar to most of the other services, WebCenter Portal provides dedicated task flows to access WebCenter Content.

In this training module you will use the Content Presenter task flow. The Content Presenter task flow allows authorized users to retrieve content from WebCenter Content and display it in a WebCenter Portal application.

This task flow uses a display template to render the content. Although WebCenter Portal comes with a set of predefined content presenter templates, you can create your own templates that suit the way you want to present your documents.

In this training module, following our examples, you will learn how to create, package, and use such content presenter templates. The examples are themed for our demo web application, El Piju. We will show you how to enhance the user interface of this application with content presenter task flows and custom templates.

If you want to download the demo application and its content, see the *Related Documents* page of this training module.

The Content Presenter templates are well documented in *Chapter 29, Creating Content Presenter Display Templates* of the book *Oracle® Fusion Middleware Developer's Guide for Oracle WebCenter Portal*. Content Presenter task flow is available since Oracle WebCenter Portal 11.1.1.4.0 version, but some of the features illustrated in this training will work only in Oracle WebCenter Portal 11.1.1.6.0.

Note: Although this training module discusses content presenter task flows and display templates in the context of a custom WebCenter Portal application, the same display templates can be made available as portal resources and used at run time in Oracle WebCenter Portal's Spaces.

Prerequisites

The training module assumes that you are familiar with creating custom WebCenter Portal applications, using Oracle JDeveloper. For the first template, we will show detailed steps of how to create the template, pack it as a portal resource, and use it in an application. For the second example template, we will show and explain the template source, but will not show all the steps in details.

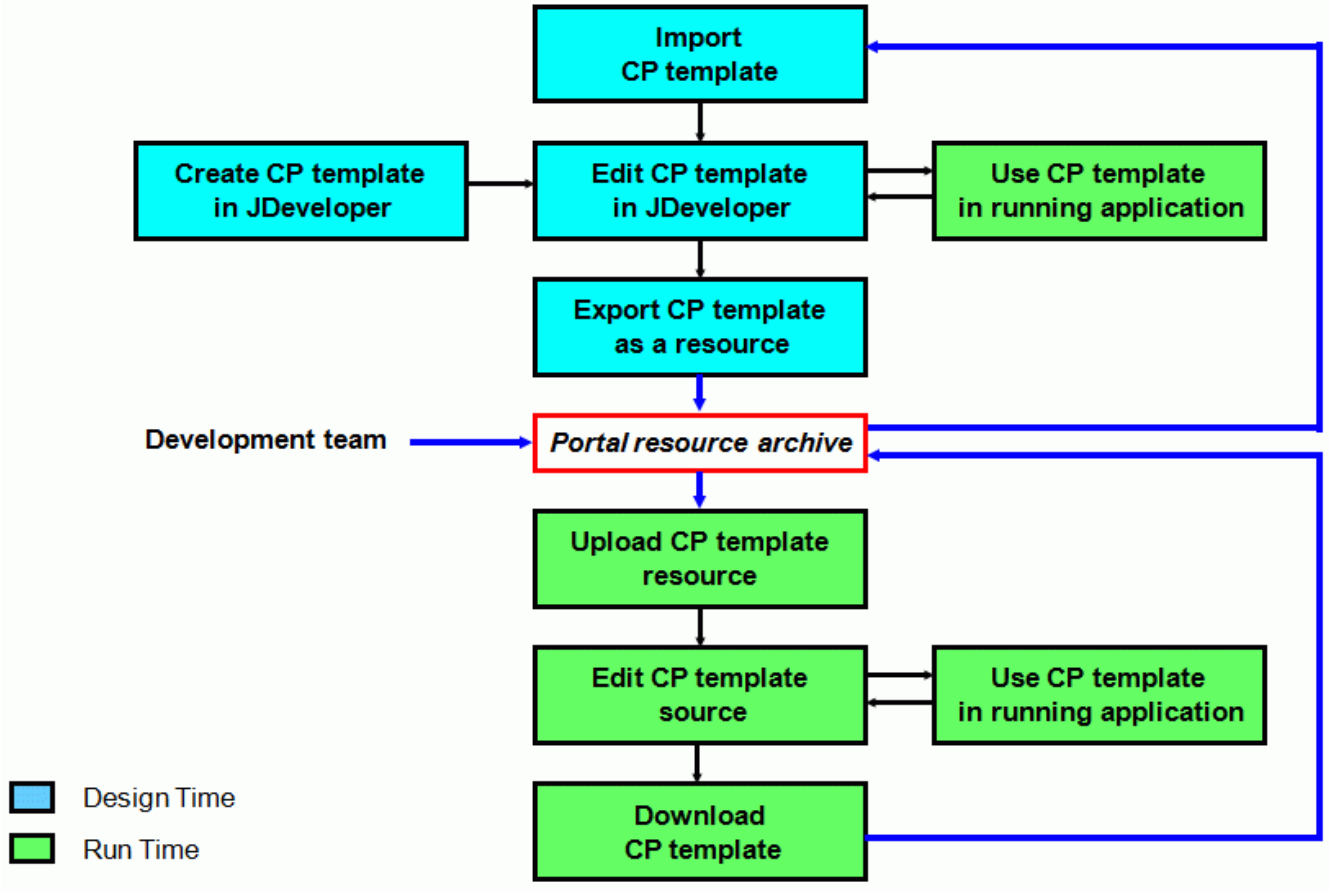
Although all templates are created as a supplement for the above mentioned El Piju demo application, you can test them without this application. All you need is Oracle JDeveloper 11.1.1.6.0, a WebCenter Content Server instance, and some images in the server. To download the demo content used in El Piju application, see the *Related Documents* page of this training module.

Overview of Content Presenter Templates

A few important features of the content presenter templates:

- A Content Presenter display template is a JSFF file (JSF page fragment) that defines how Content Presenter renders content items on a WebCenter Portal page.

- Display templates can use the full set of rich ADF components, so you can quickly and easily create robust and attractive templates to display your content. You are not, however, required to use these components in your template.
- A Content Presenter display template can handle a single content item, multiple content items, or combinations of the two. Display templates can call each other: a single item template may call another single item template, and a multiple item template may call either another multiple- or single item templates. For example, a multiple content item template might render tabs for each item and then call a single item template to render the details of a selected item.
- Each content item is associated with a specific content type defined in the WebCenter Content Server repository. A content type defines the properties of the content item. Content types can map to Content Server profile definitions and Site Studio Region Definitions. As a Content Presenter display template developer, you need to know the names of the properties defined for the associated content type so that you can access the value of these properties and define how to display the selected content item(s) on the page. Later in this module we will show ways, how you can discover the name of the properties.
- Content Presenter display templates are portal resources. They can be developed and used either during design- or run-time. They can be packaged into standard portal resource archives. The following diagram illustrates the lifecycle, that is, the round-trip development process, for content presenter (CP) display templates.



There are only two main differences between page template resources and content presenter template resources:

- Content Presenter templates cannot be created at run time.
- You cannot edit Content Presenter templates graphically at run time. You can only edit the display template source.

You will see the details of working with the Content Presenter template resources later in the training module.

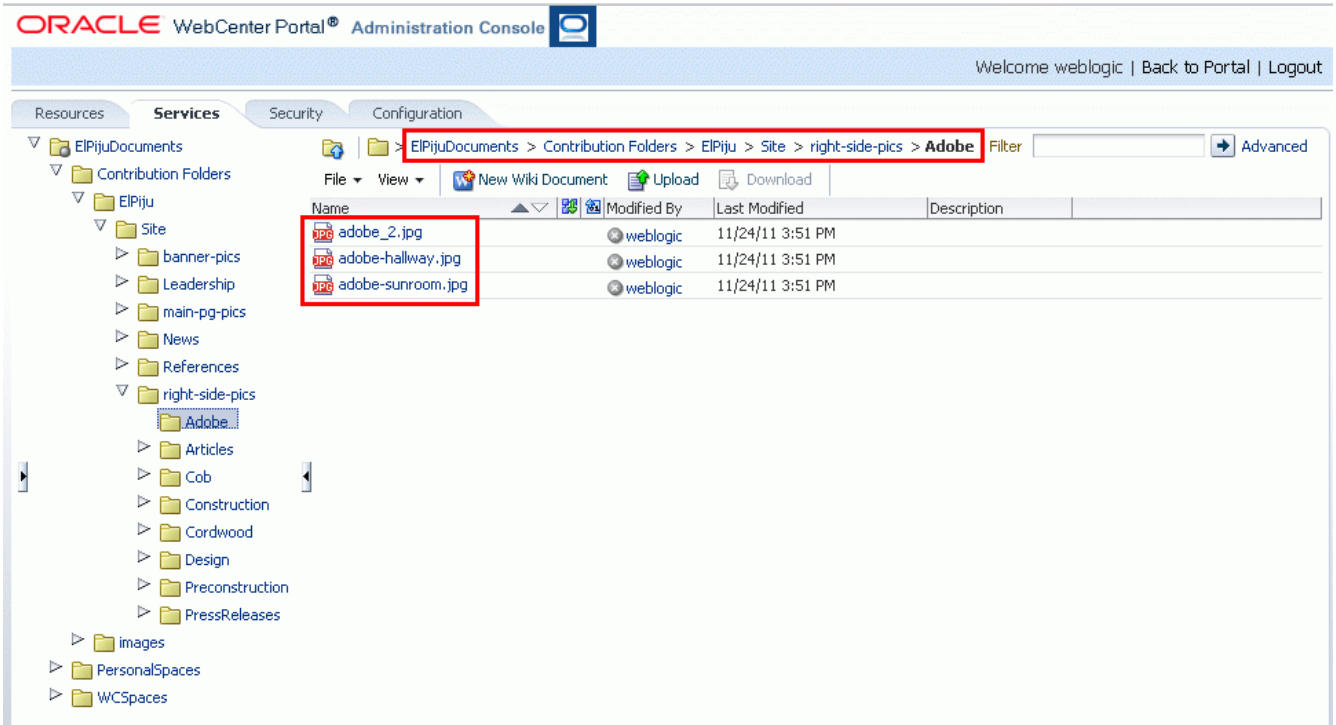
To learn more about page templates, please use our training module: *Creating and Using Page Templates in Oracle WebCenter Portal Applications*.

Create Image Sidebar Template

Here is one of the pages of our application, the Adobe page.

As a design element for these pages, we use images related to the actual page content on the right side of the page. Note that the red border is not displayed on the page; it is used to highlight the part of the screen shot with the image bar.

These images are stored in the content server, under a folder related to the actual page where the images are used. Here you can see – using the El Piju application's Administration Console – the folder structure of the application's content in the repository. All 3 images are stored in the same folder: `Contribution Folders > El Piju > Site > right-side-pics > Adobe`.



If you add the images directly into the page, here is how the resulting page source fragment will look:.

```
<cust:panelCustomizable ...>
  <cust:showDetailFrame ...>
    <af:image id="image1"
              source="/content/ ... /Adobe/adobe-hallway.jpg"/>
    <af:image id="image2"
              source="/content/ ... /Adobe/adobe-sunroom.jpg"/>
    <af:image id="image3"
              source="/content/ ... /Adobe/adobe_2.jpg"/>
   </cust:showDetailFrame>
</cust:panelCustomizable>
```

Note: In the source code above we abbreviated the path to the actual images.

It is a tedious process to add all the images individually. The resulting page is also difficult to change; if we want to change the number or the name of the images, the page source has to be modified.

In the following steps, we replace the three images with an invocation of the Content Presenter task flow. This task flow will use a custom Content Presenter display template to render all the images under a specific folder as a vertical image bar. Obviously the template will display multiple content items.

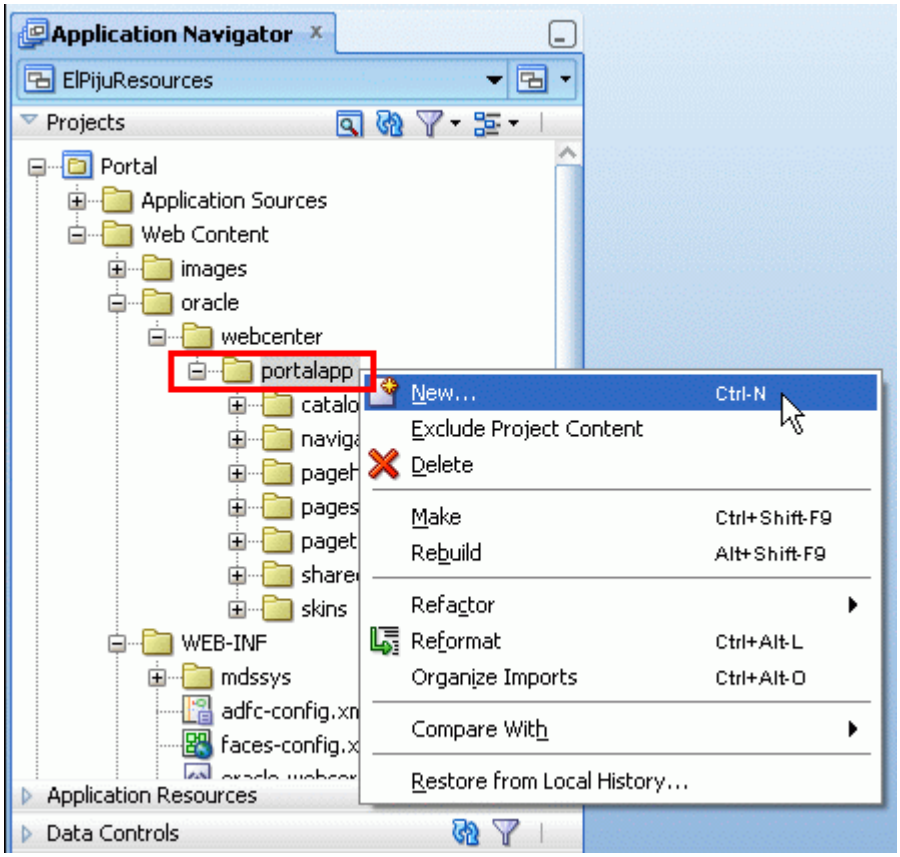
1. Create a new WebCenter Portal application for the Content Presenter templates.

   Although we could use any application, even El Piju itself to develop the Content Presenter template and other portal resources, we recommend that you create a separate application, for example one called `ElPijuResources`, which can hold several resources, including Content Presenter templates.

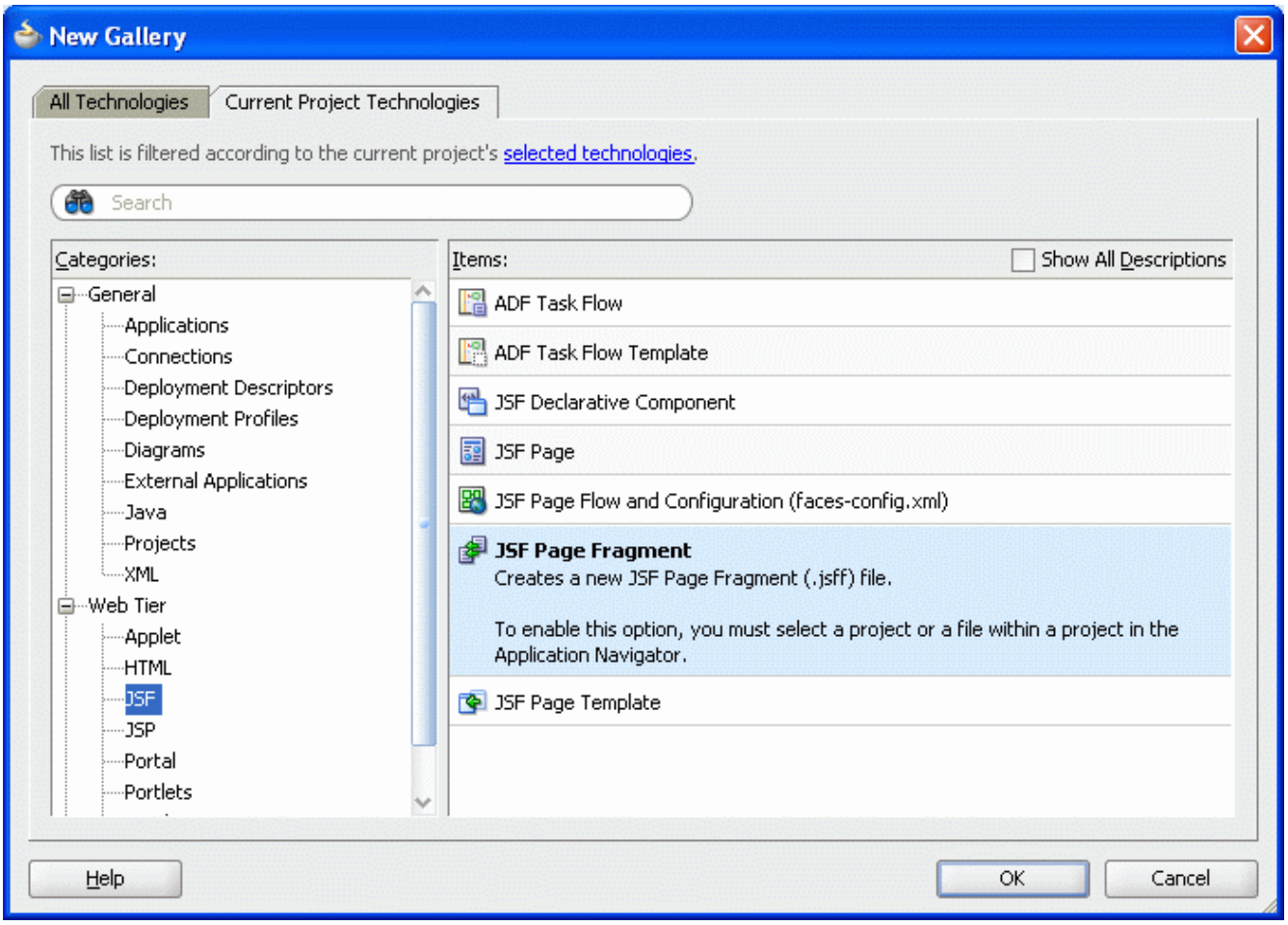
2. Create a new JSF page fragment for the template.

Since the Content Presenter template will become a portal resource, the file itself must reside somewhere under the `oracle/webcenter/portalapp` folder. We recommend that you create a separate folder, for example **`oracle/webcenter/portalapp/contenttemplates`** and create all the JSF page fragments there.

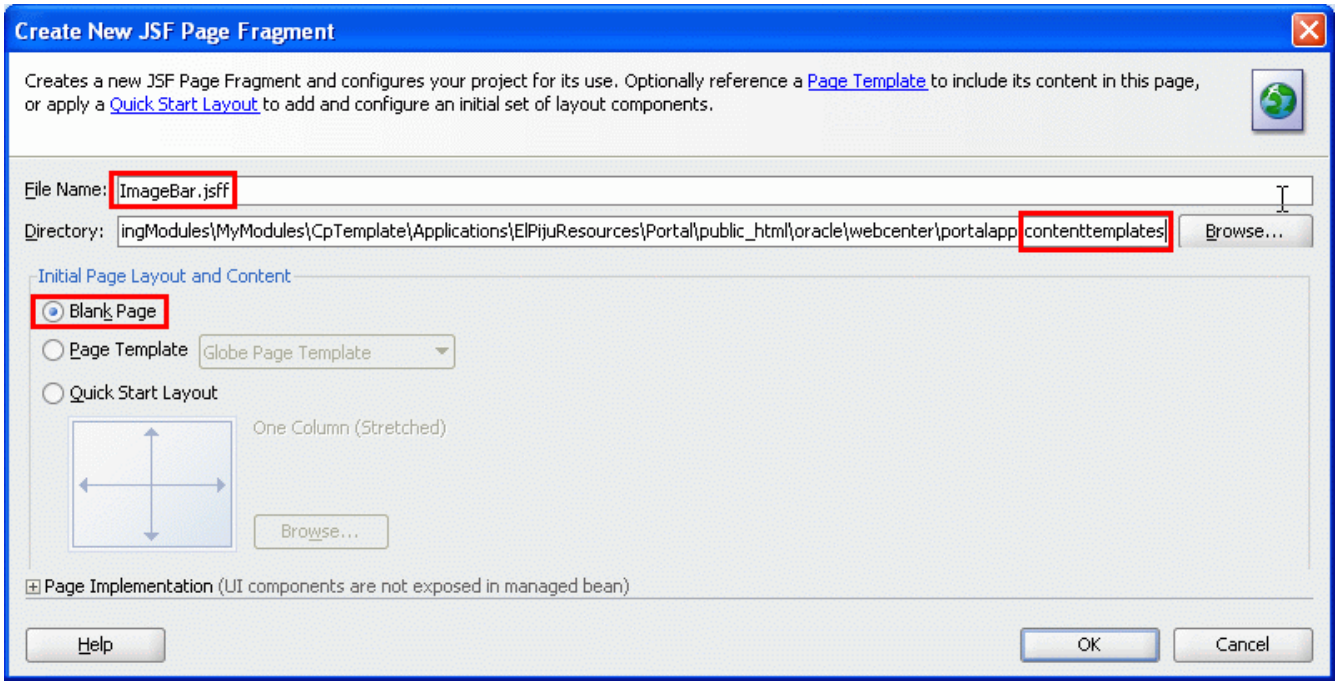
a. In the Application Navigator pane, expand the nodes under `Web Content`, then right click the `portalapp` node and select **New** from the pop-up menu.



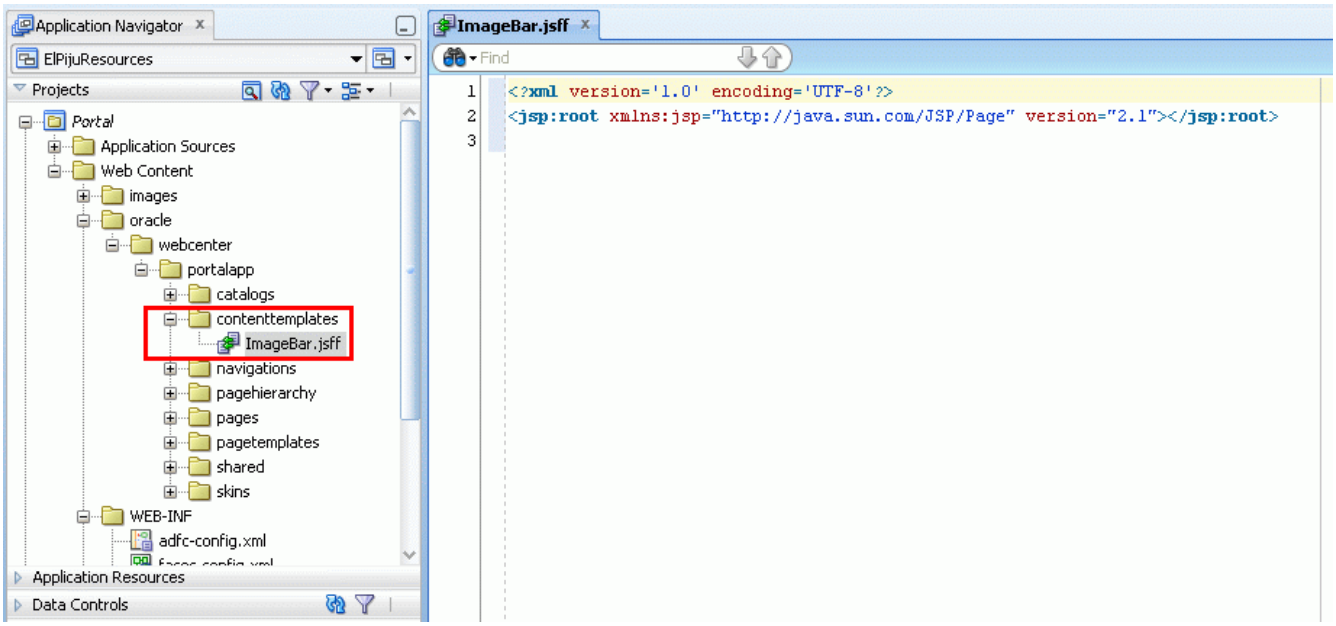
b. Select **JSF** from the Web Tier categories and choose **JSF Page Fragment**. Click **OK**.



c. In the Create new JSF Page Fragment window, set the File Name field to **`Imagebar.jsff`**, add **`\contenttemplates`** to the end of the Directory field, and make sure that **Blank Page** is selected. Click **OK**.



d. A new file, `ImageBar.jsff` is created under the `contenttemplates` folder. In the Editor pane, click the **Source** tab; you will see an empty JSF page fragment.

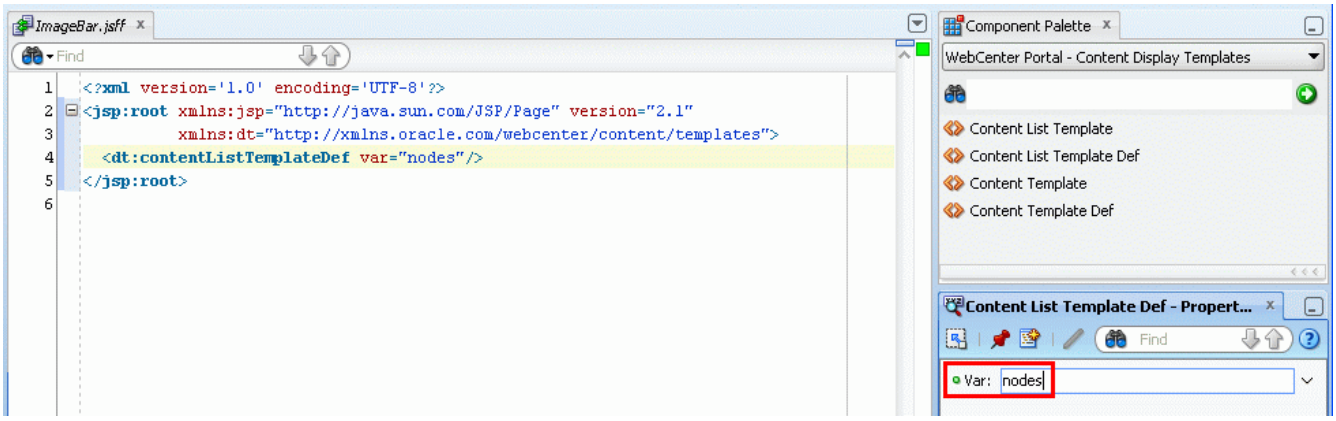


3. Add the Content Presenter template definition.

a. Select WebCenter Portal – Content Display Templates category in the Components palette. Since we are developing a multiple item template, drop **Content List Template Def** tag to the page.

b. The newly added `<dt:contentListTemplate ...>` element must have an attribute called `var`, that will define the variable which holds the collection of the items that has to be rendered by the template.

Place the cursor inside the element and, in the Property Inspector pane, you can set the value of the `var` attribute, for example, to `nodes`.



This is the resulting page fragment. Note that you could add the required attribute in the source window as well.

```
<?xml version='1.0' encoding='UTF-8'?>
<jsp:root
     xmlns:jsp="http://java.sun.com/JSP/Page" version="2.1"
     xmlns:dt="http://xmlns.oracle.com/webcenter/content/templates">
  <dt:contentListTemplateDef var="nodes"/>
</jsp:root>
```

4. Populate the template body.

   Now you have to add tags to the template definition's body that will render the items. It is beyond the scope of this training module to discuss the details of ADF layout management and user interface tags.

   You can drop the necessary tags from the Component Palette's ADF Faces group or enter the source code.

   - We need a container that arranges the images vertically, one below the other. Use Panel Group Layout with vertical layout style.
   - Inside the container we need to process all of the items (images). Use the `<af:iterator>` tag.
   - Inside the loop use the `<af:image>` tag to display the image.

   Here is the resulting source code:

```
01 <?xml version='1.0' encoding='UTF-8'?>
02 <jsp:root
03    xmlns:jsp="http://java.sun.com/JSP/Page" version="2.1"
04    xmlns:dt="http://xmlns.oracle.com/webcenter/content/templates"
05    xmlns:af="http://xmlns.oracle.com/adf/faces/rich">
06   <dt:contentListTemplateDef var="nodes">
07     <af:panelGroupLayout id="pgl1" layout="vertical" valign="top">
08       <af:iterator id="i1" rows="0" var="node" value="#{nodes}">
09         <af:image id="i2" source="#{node.url.renderUrl}"/>
10       </af:iterator>
11     </af:panelGroupLayout>
12   </dt:contentListTemplateDef>
13 </jsp:root>
```

   A few notes:

   Line 05

   The `af` namespace has to be added to the page since it contains the ADF Faces tags.

   Line 07

   The Panel Group Layout will arrange its content vertically.

   Line 08

   The iterator processes all the items (`value="#{nodes}"`) and place the current item into the variable called node. The `rows="0"` attribute tells the iterator to fetch all of the available items, even if we don't expect more than 3 images in the folder.

Line 09

The image tag needs the source attribute, a URL, where the current images can be fetched. The following EL expression provides access to the current item:

```
source="#{node.url.renderUrl}"
```

Soon we will discuss the content item attributes in details.

5. Enhance the template.

Although the template will display the images correctly, we could make the code more robust . Here is an example of enhanced source code with explanations:

```
01 <?xml version='1.0' encoding='UTF-8'?>
02 <jsp:root
03       xmlns:jsp="http://java.sun.com/JSP/Page" version="2.1"
04       xmlns:dt="http://xmlns.oracle.com/webcenter/content/templates"
05       xmlns:af="http://xmlns.oracle.com/adf/faces/rich">
06   <dt:contentListTemplateDef var="nodes">
07     <af:panelGroupLayout id="pgl1"
08         layout="vertical" valign="top"
09         rendered="#{not empty nodes}">
10       <af:iterator id="i1" rows="0" var="node" value="#{nodes}">
11         <af:image id="i2" source="#{node.url.renderUrl}"
12                   inlineStyle="margin-bottom:5px; width:194px;"
13                   rendered="#{(not empty node.primaryProperty) and
14                               node.primaryProperty.isImage}"/>
15       </af:iterator>
16     </af:panelGroupLayout>
17   </dt:contentListTemplateDef>
18 </jsp:root>
```

Line 09

We will render the Panel Group Layout only if the Content Presenter task flow was called with a non-empty collection of items. We could add a warning message, but it is not implemented here.

Line 12

We add a 5 pixel wide margin to the bottom of all images, thus separating the images. We also know that in our application template the column for the right-side images is 194 pixels wide. Here we resize the images to fit this width.

Line 13 and 14

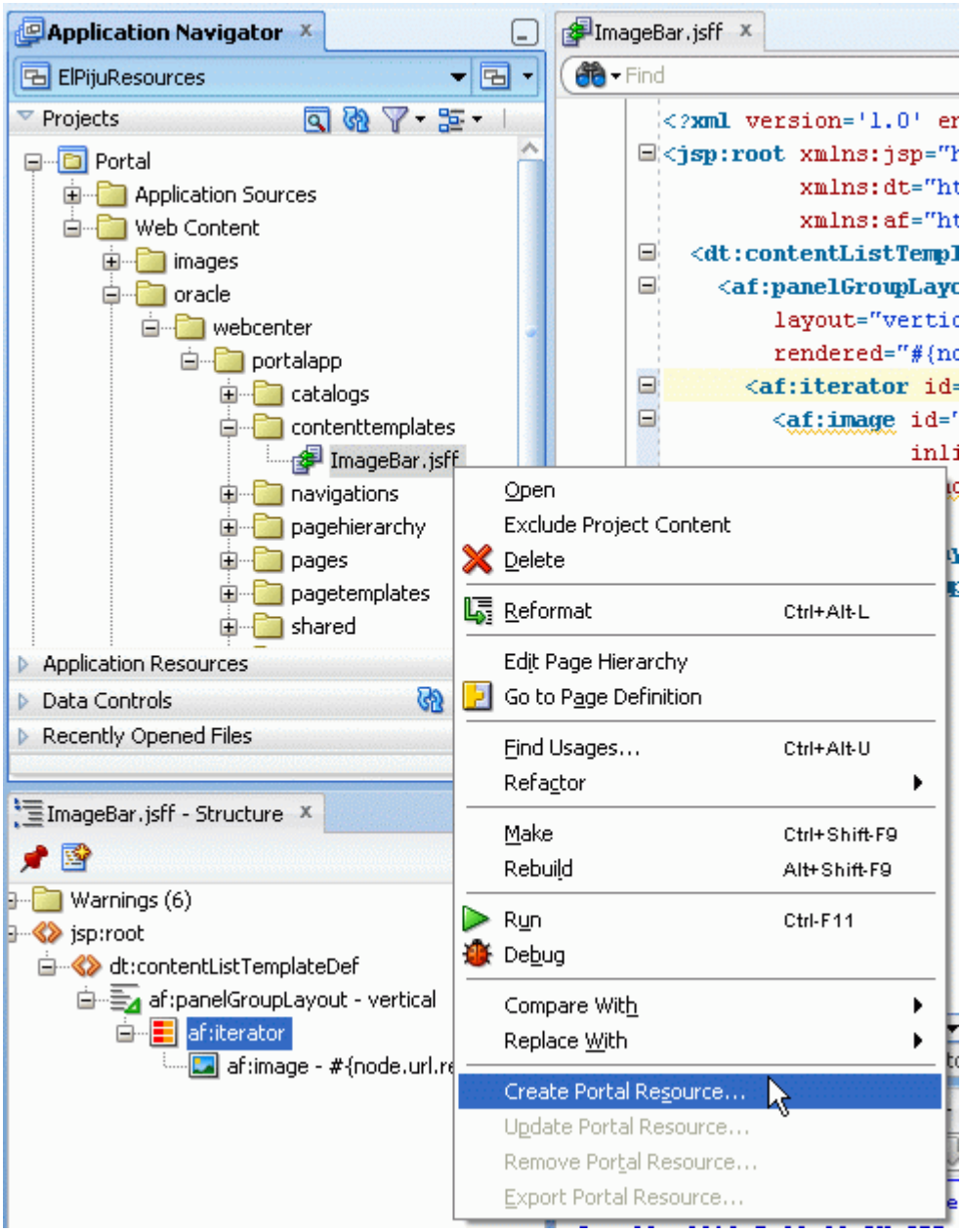
We will render only images from the collection. If an item is not of image type, we will silently ignore it. In the next example you will see more about using the item's properties, like `primaryProperty`, or `isImage`.

6. Create a portal resource.

Once we have the page fragment, we can create a portal resource from it. Creating a portal resource will store the page fragment in MDS (metadata storage) and associate a name and resource type with it.

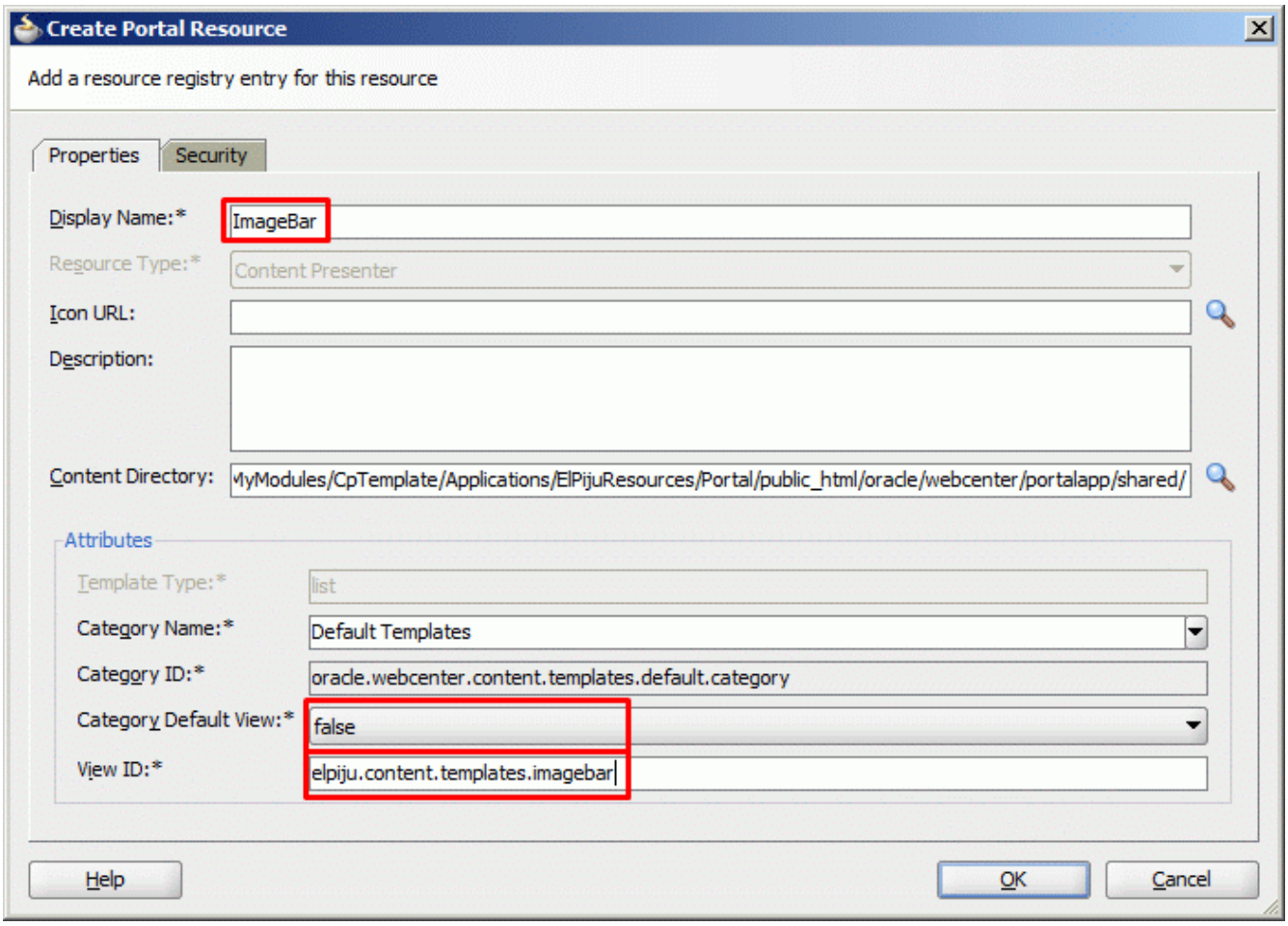
Later in the module, we will pack the resource into a portal resource archive that can be imported into other WebCenter Portal applications.

a. Select `ImageBar.jsff` in the Application Navigator. Right-click the file name and choose **Create Portal Resource** from the pop-up menu.



b. The Create Portal Resource window opens. You can see from the gray Resource Type field that the wizard deduced that you are creating a resource from a Content Presenter template. Give the template a descriptive name, for example: **`Image Bar`**. Change the Category Default View to `false`, since this is not the default template. Define a View ID, for example: **`elpiju.content.templates.imagebar`**.

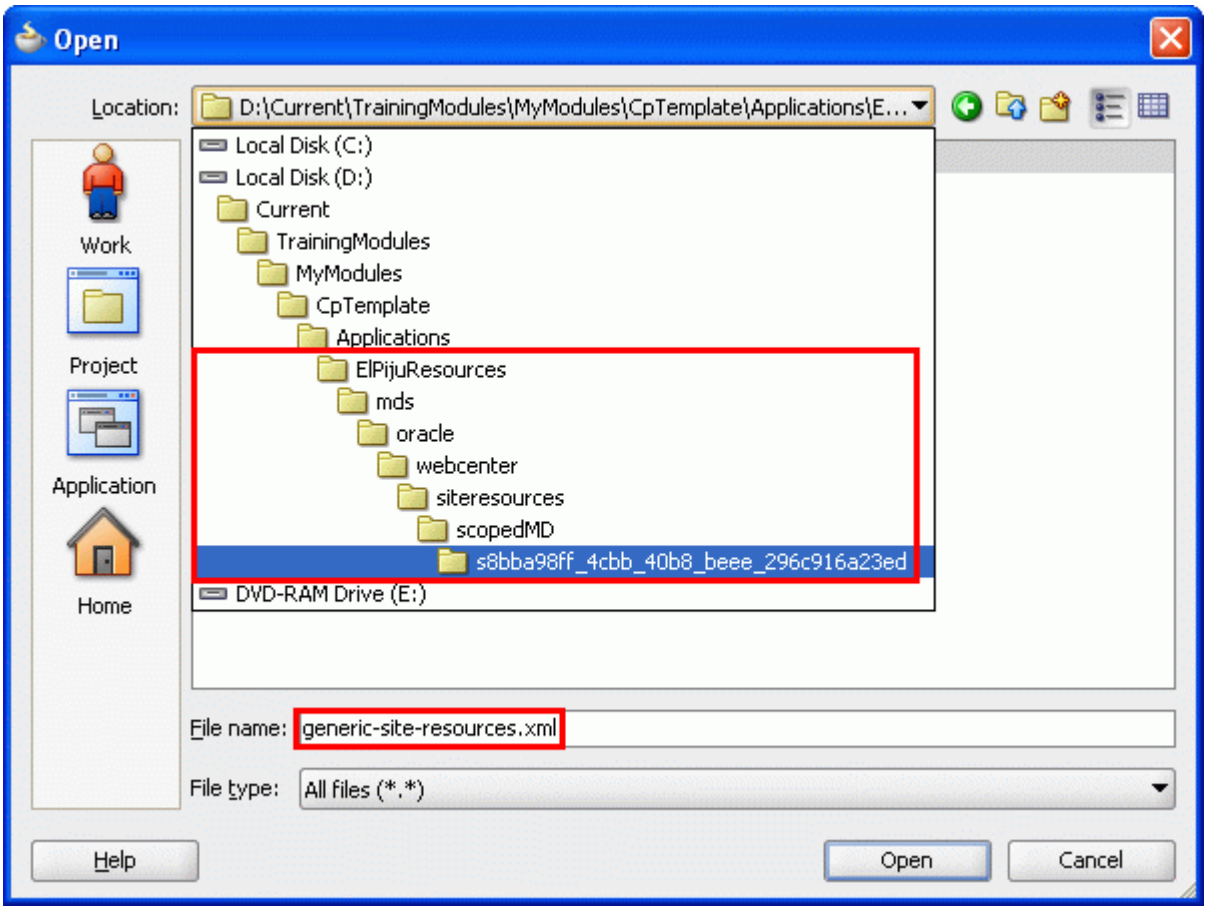
   You can leave the other fields with their default values. You could add an icon and a description text, but it is not required. Content Presenter templates can be grouped into categories, but since we are not planning to create too many custom templates, we will place all of them into the default category.



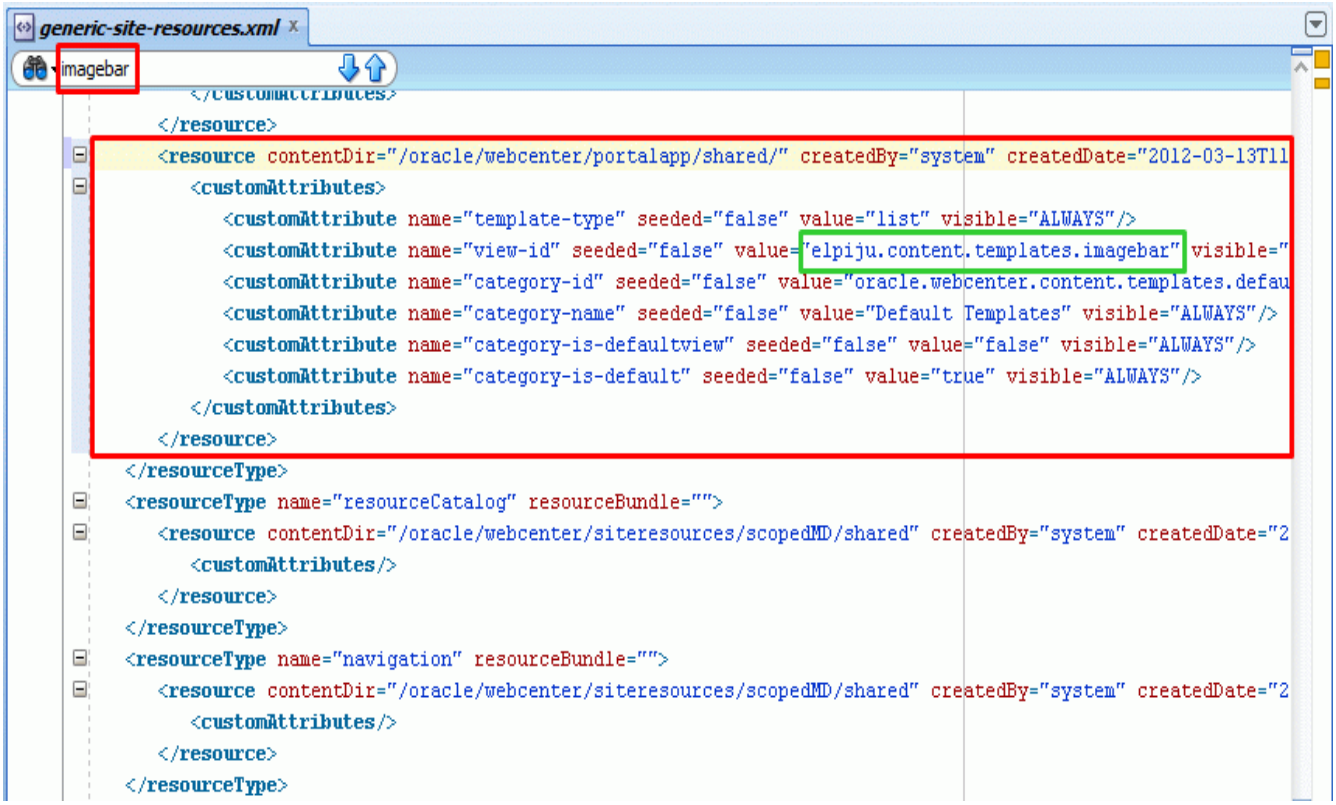
c. Set the resource visibility to true.

The portal resource will be stored in MDS. During development, MDS resides in a folder called `mds` under the application's root folder.

Use Open in the File menu, then select the file **`generic-site-resources.xml`** from the folder `.../ElPijuResources/mds/oracle/webcenter/siteresources/scopedMD/<unique_ID>`



d. In the file, find the resource, for example search for **`imagebar`**.



In the `<resource ...>` tag's line, scroll to the right and change the visible attribute to true, i.e. **`visible="TRUE"`**.

If you forget to change this attribute, when you test the template in design time, the task flow will not find the template, even if you correctly refer to its view ID.

Save the modified file.

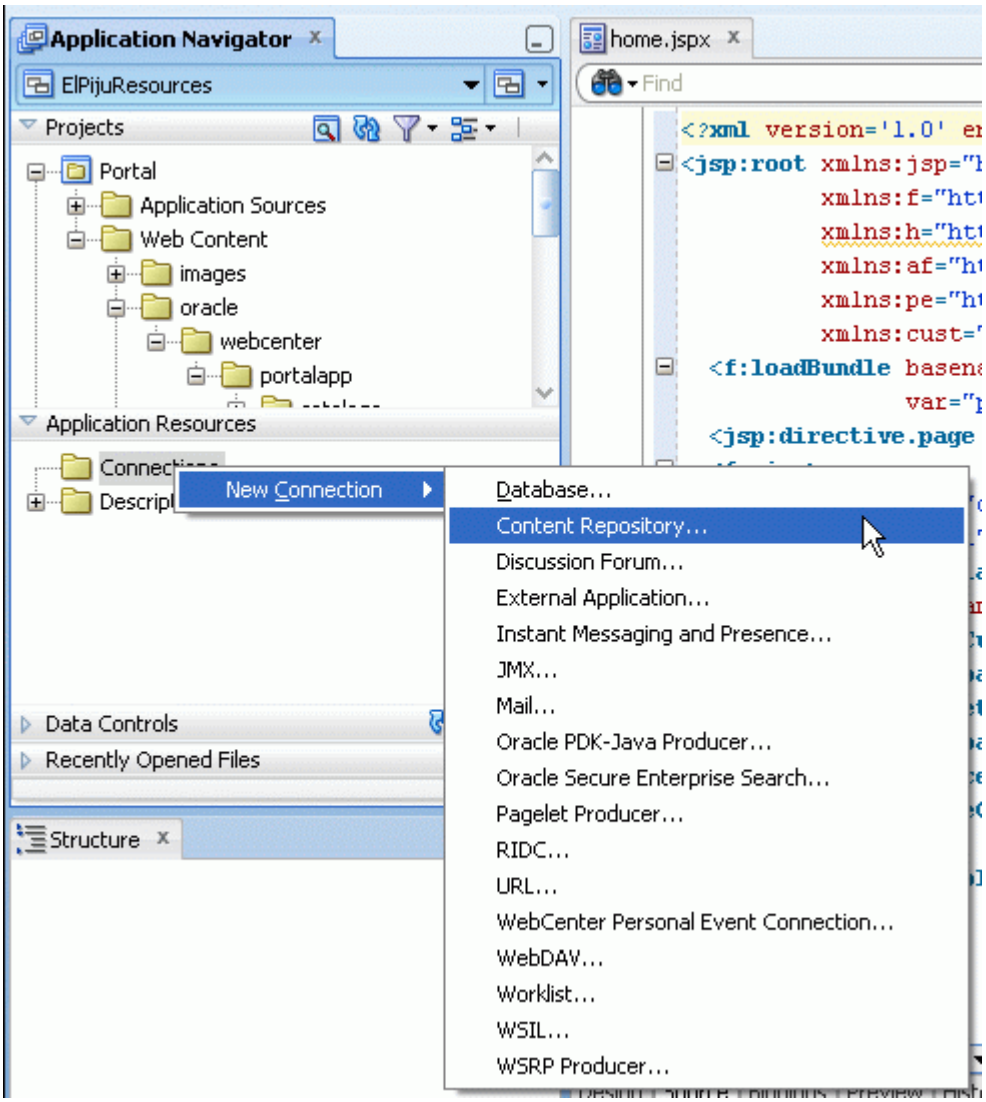
Test the New Template

This Content Presenter template is a portal resource; it is meant to be packed up, then imported to another application (El Piju for example), where it will be used. But you can also test the template without packing, and importing, in the current application. It is not a full test, since this application does not necessarily have all the components, like page template and pages, but still you can see how the template renders the content.

7. Create a Content Repository connection.

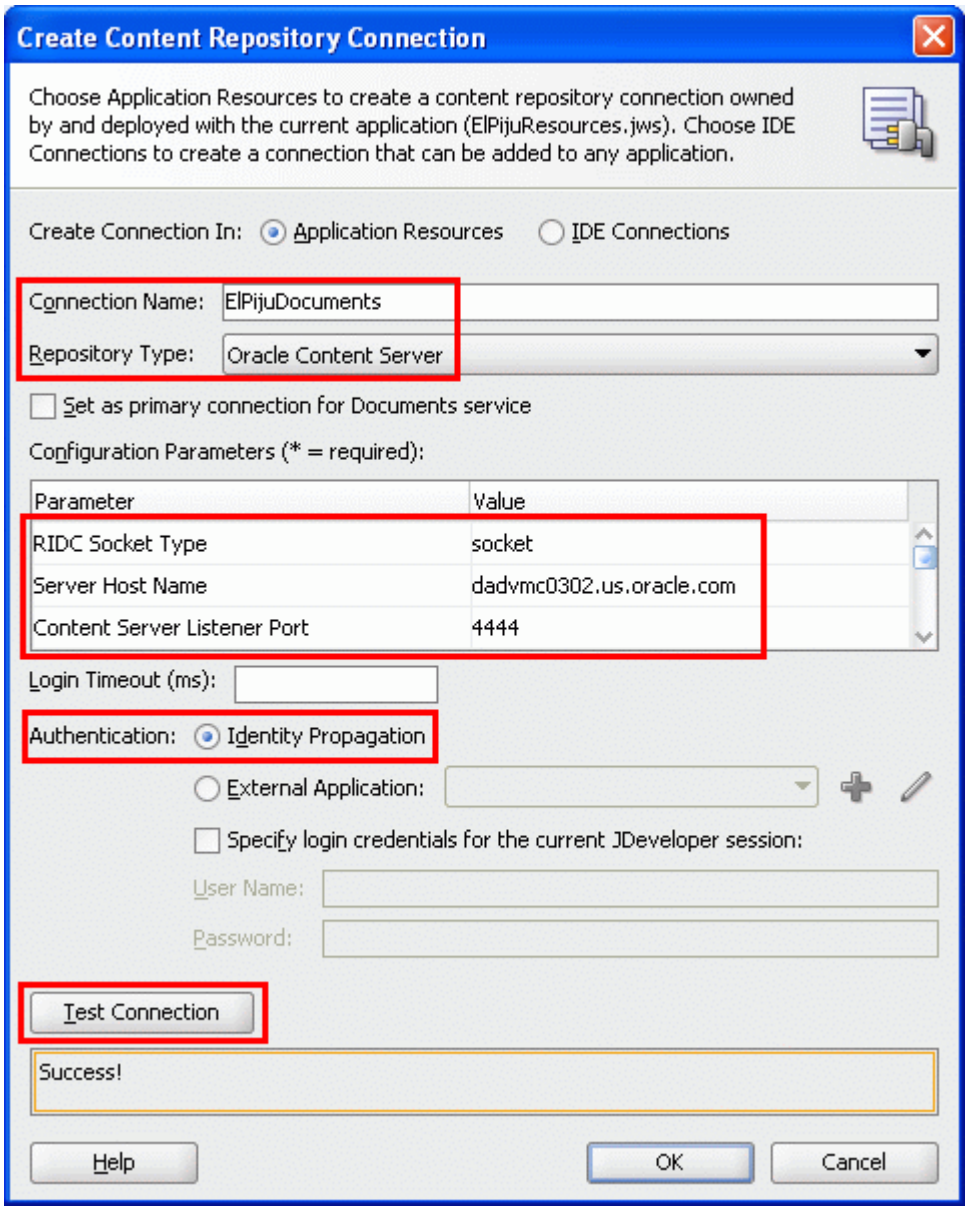
   You need a running Oracle WebCenter Content Server instance and a connection to it in order to test the Content Presenter template.

   a. In the Application Navigator pane, right-click Connections under Application Resources and select **New Connection > Content Repository...**



b. In the Create Content Repository Connection window, create a connection in the application resources with the following attributes:

Connection Name	unique name, for example `ElPijuDocuments`
Repository Type	**Oracle Content Server**
RDIC Socket Type	**socket**
Server Host Name	*<content server name or IP>*
Content Server Listener Port	*<content server listener port>* default: **4444**
Authentication	**Identity Propagation**



Test the connection; make sure it successfully connects to the Content Server.

8. Add a Content Presenter task flow with the new template to the Home page.

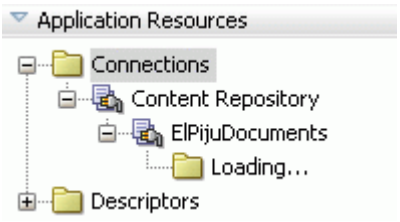
   a. Open the **Web Content > oracle > webcenter > portalapp > pages > home.jspx** page in Source mode.

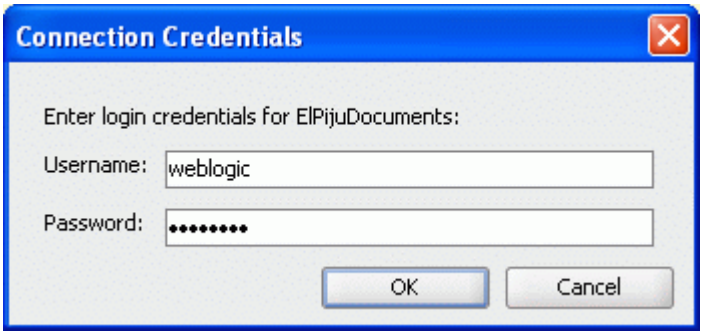
   There are two ways to add a Content Presenter task flow to the page:

   - Add it from the resource palette then configure the task flow parameters.
   - Drop a content item to the page and configure the template.

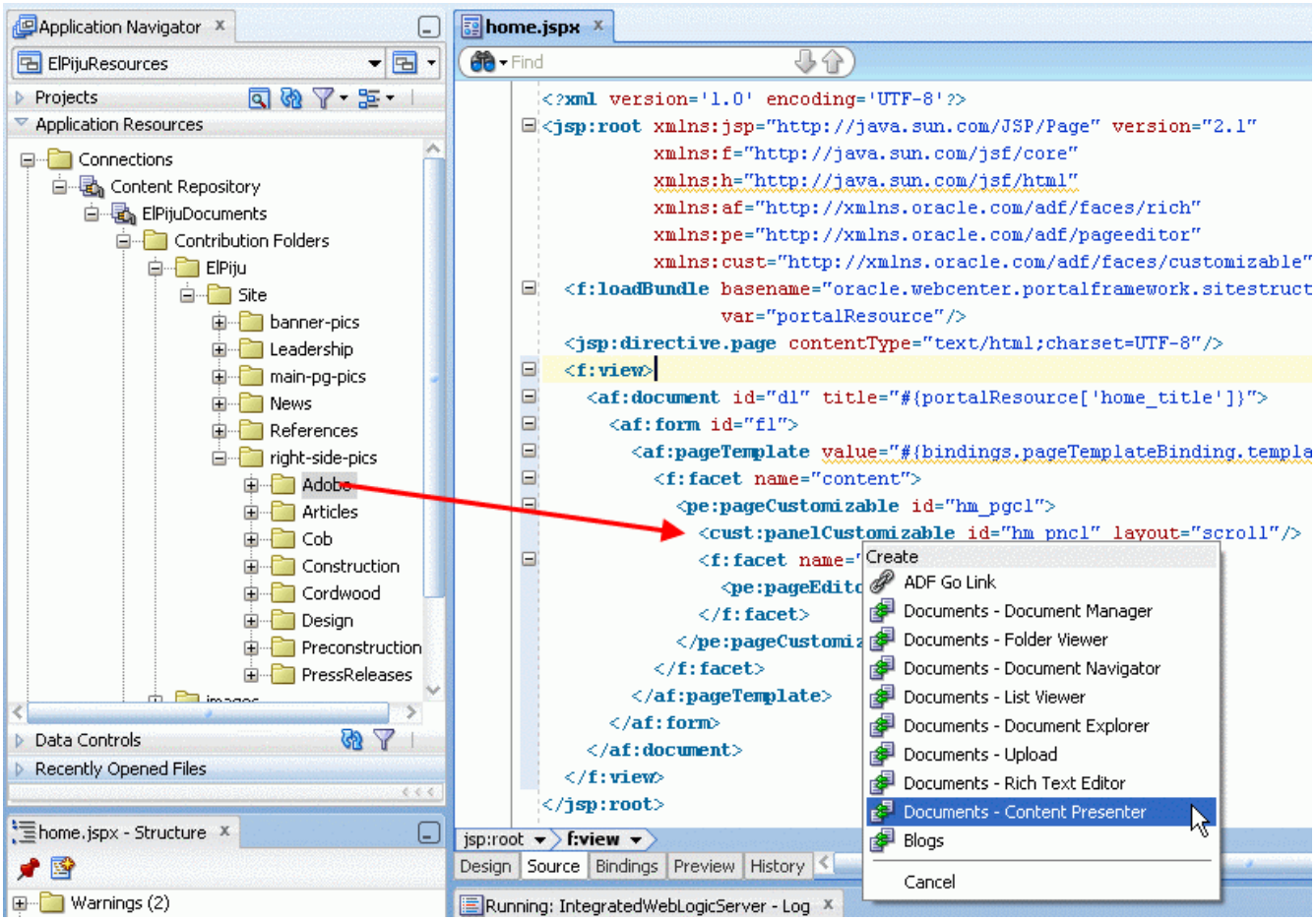
   Since the second option is much simpler, let's use it.

   b. Expand the Content Repository connection in Application Resources. You have to provide credentials to access the Content Server.



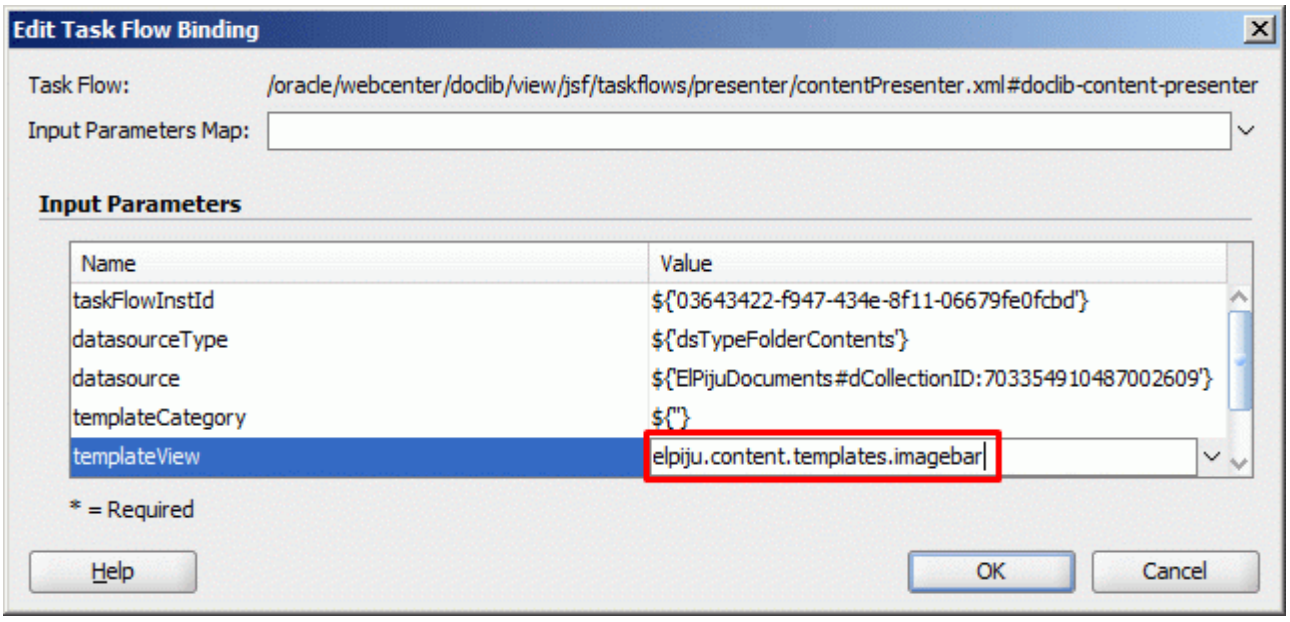


c. Drill down to a folder that contains the images to be displayed in the sidebar. In our case it is: **Contribution Folders > ElPiju > Site > right-side-pics**. Drag the folder, **Adobe** and drop it onto the Panel Customizable component. From the Create pop-up, choose **Documents – Content Presenter**.



d. In the Edit Task Flow Binding window, you can see the advantages of this method: most of the task flow input parameters are already filled in. The parameter `datasourceType` indicates that the task flow will display the content of a folder (`dsTypeFolderContent`) and the `datasource` parameter points to the actual folder, using the folder's unique ID in the Content Server.

You only have to provide the template's ID in the `templateView` parameter: **`elpiju.content.templates.imagebar`**.



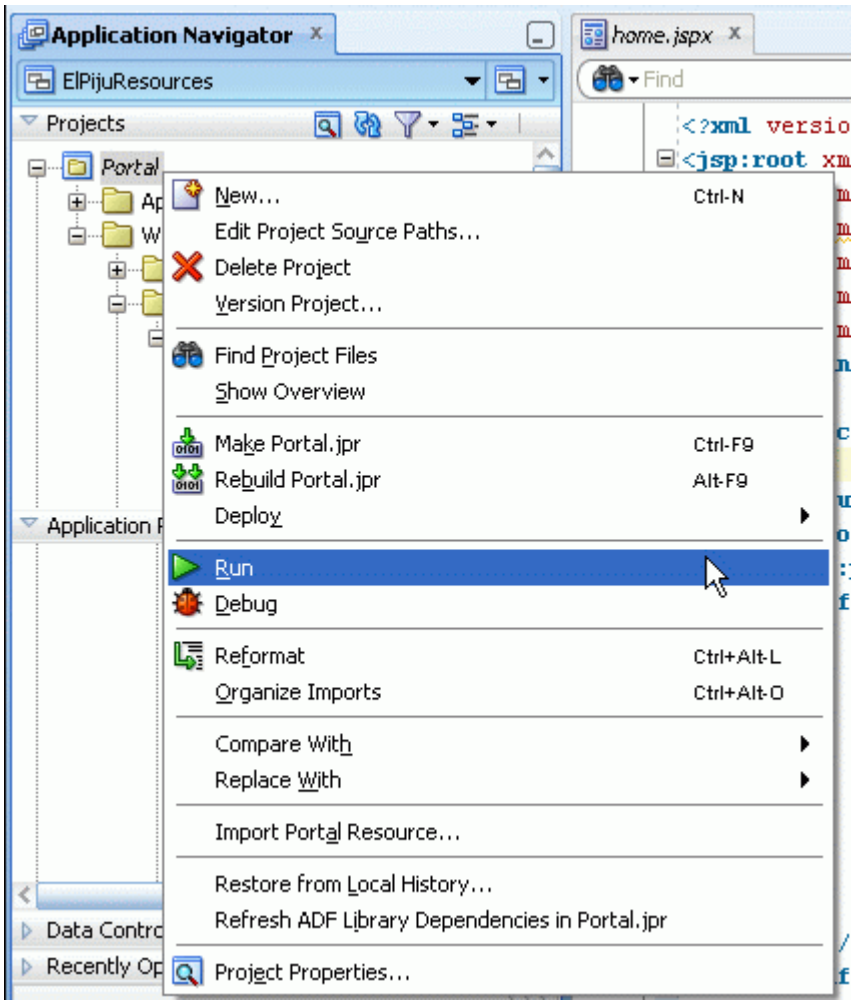
As we mentioned above, you can also add the task flow directly to the page. It is usually more complicated because you have to manually set the above parameters and that often requires finding the unique document- or collection ID from the Content Server. But this offers more possibilities, especially when using multi-item templates. In addition to displaying a folder's content, you could display

- a list of individual items.
- the result of a CMIS search.
- the result of a personalization scenario.

To learn more about the task flow parameters refer to *Chapter 31.7.1 Content Presenter Task Flow Parameters and Out-of-the-Box Display Templates* in the *Oracle® Fusion Middleware Developer's Guide for Oracle WebCenter*.

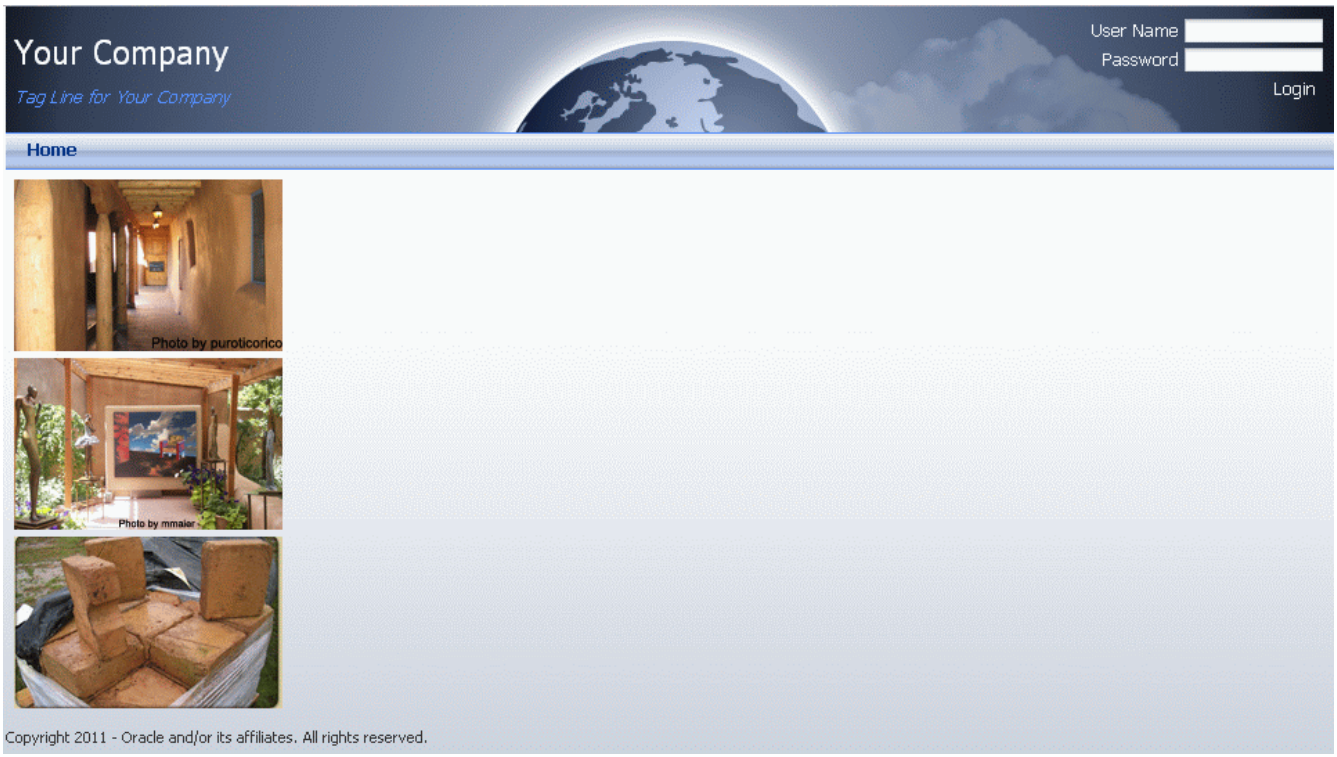
9. Run the application.

   a. Select the Portal project. Right-click on the project and choose **Run** from the pop-up menu.



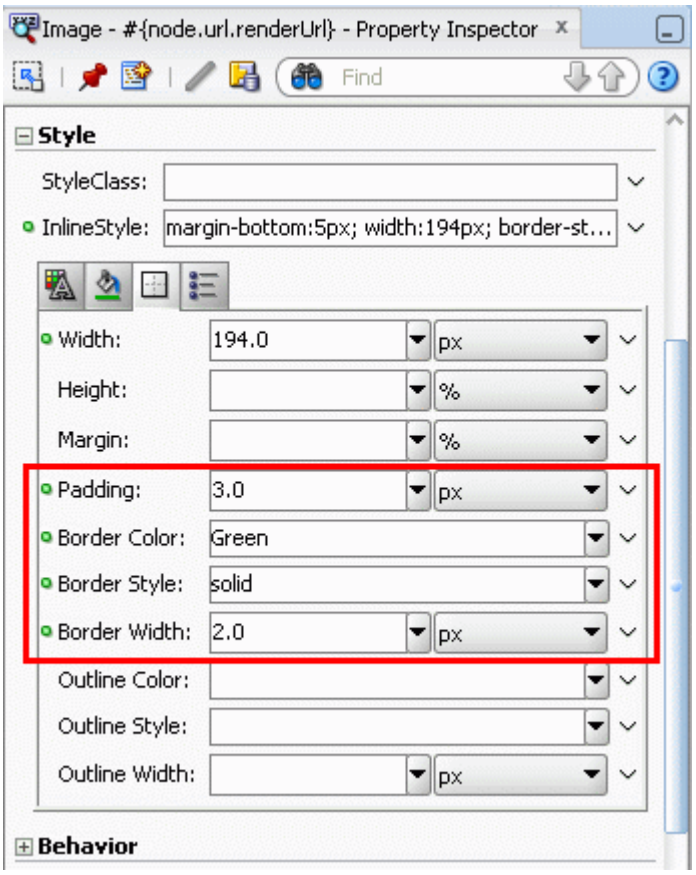
- JDeveloper compiles the application, including the pages and the Content Presenter template.
- JDeveloper will start the embedded IntegratedWebLogicServer WLS server.
- JDeveloper deploys the current application to the WLS server
- JDeveloper will start the default browser and forces the browser to fetch the application's root URL: `http://127.0.0.1:7101/<context root>`. The browser will execute several redirections, and eventually will fetch and display the Home page.

b. The Home page should display the images from the Adobe folder as a vertical image bar.



c. The real advantage of testing the Content Presenter template in the same application is that you can quickly modify the template and test how it works.

   For example, open the `ImageBar.jsff` file again, place the cursor inside the `<af:image ...>` tag, and in the Property Inspector add some style definition, like on this screen shot:



This will add the following settings to the already existing `inlineStyle` attribute:

`inlineStyle="... border-style:solid; border-width:2.0px; border-color:Green; padding:3px;"`

You don't have to restart the application. Simply save the modified file, switch to the browser window and refresh the page. This should display the images with a green border. Similarly most of the modifications do not require restarting the application. Just save the modified files and refresh the browser page.

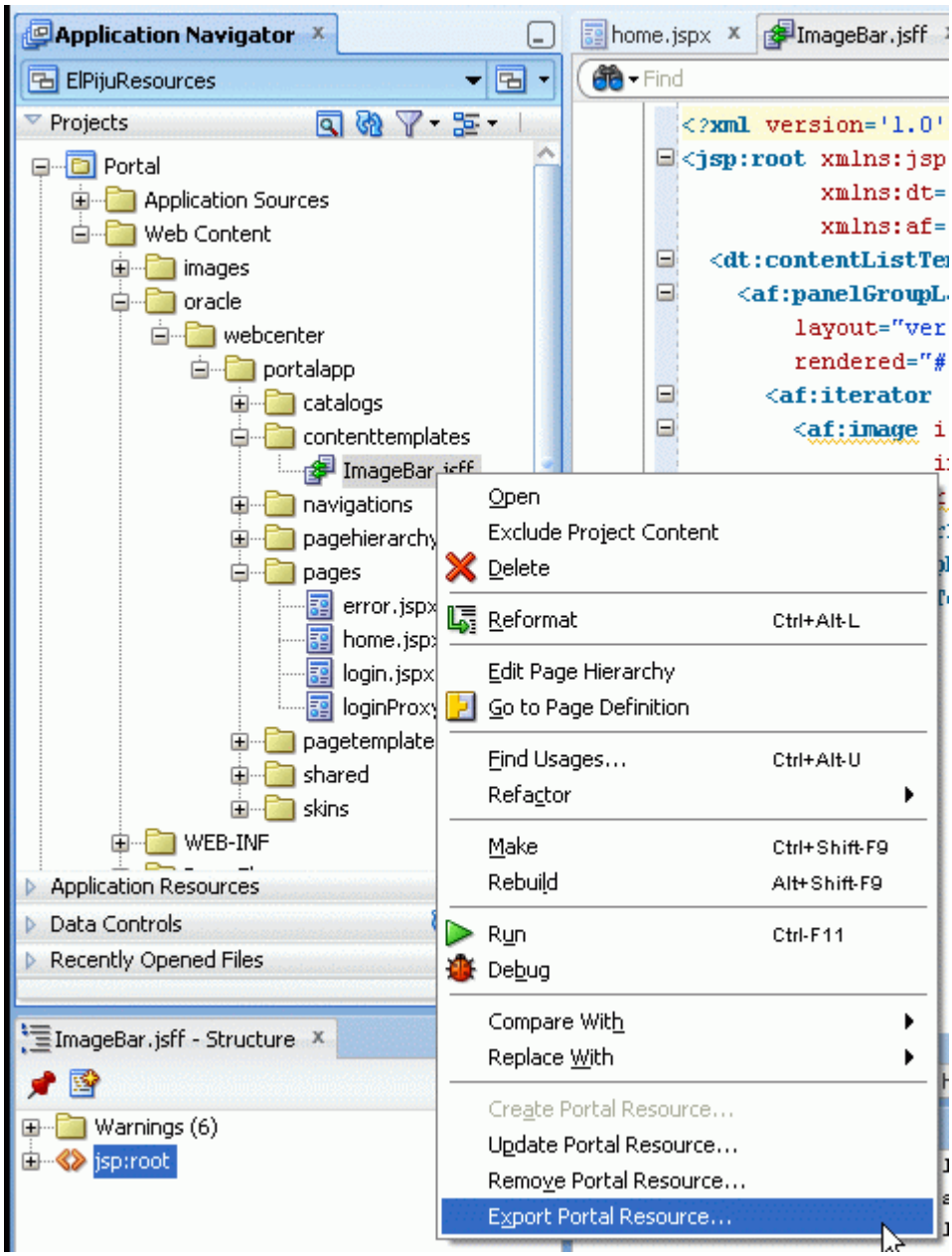
d. Finally you may remove the border settings or any other modifications you did for testing the template. We don't need them in the final template.

Work with the Portal Resource in Design Time

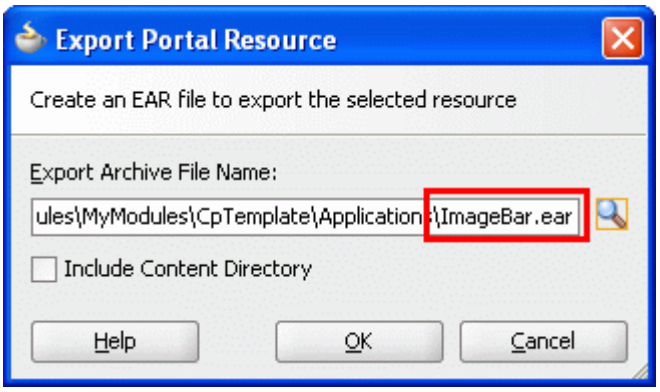
Next you are going to export the Content Presenter template as a portal resource archive and import it in JDeveloper to another application.

10. Export the portal resource.

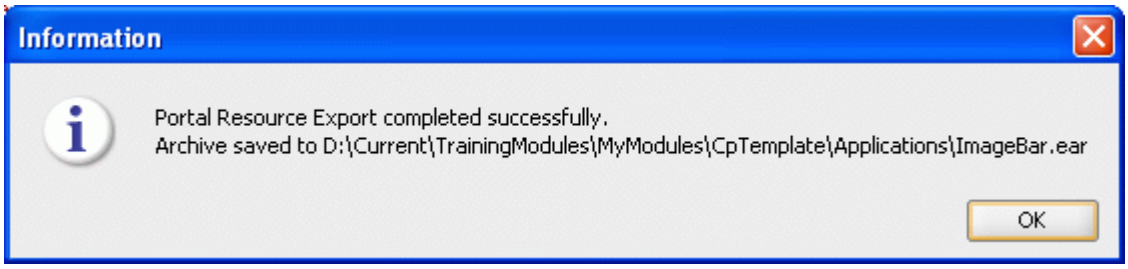
a. In the Application Navigator right-click `ImageBar.jsff` and choose **Export Portal Resource** from the pop-up menu.



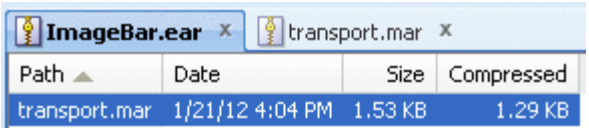
b. In the Export Portal Resource window, provide a path and file name for the resource file. We recommend that you use `ear` extension, for example: **`ImageBar.ear`**. Since this template does not use any additional file(s), you do not have to check the Include Content Directory checkbox.



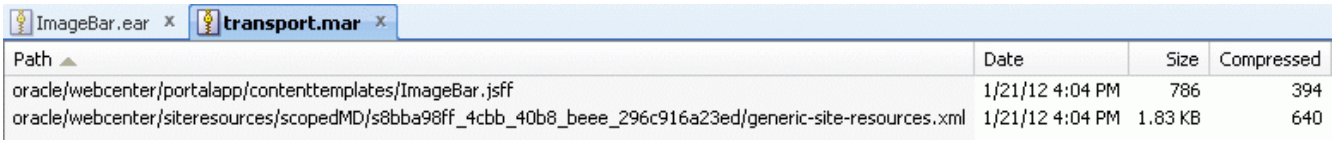
Click **OK**. You will get a confirmation window.



The portal archive is in ZIP format. If you open `ImageBar.ear` with a program that can browse ZIP archives - for example with JDeveloper -, you discover that it contains a single file, `transport.mar`.



The `mar` extension stands for "MDS archive". This file is also in ZIP format. Opening the file, you will see the two files, in their corresponding folders, that make up the Content Presenter template: `ImageBar.jsff` and `generic-site-resources.xml`.

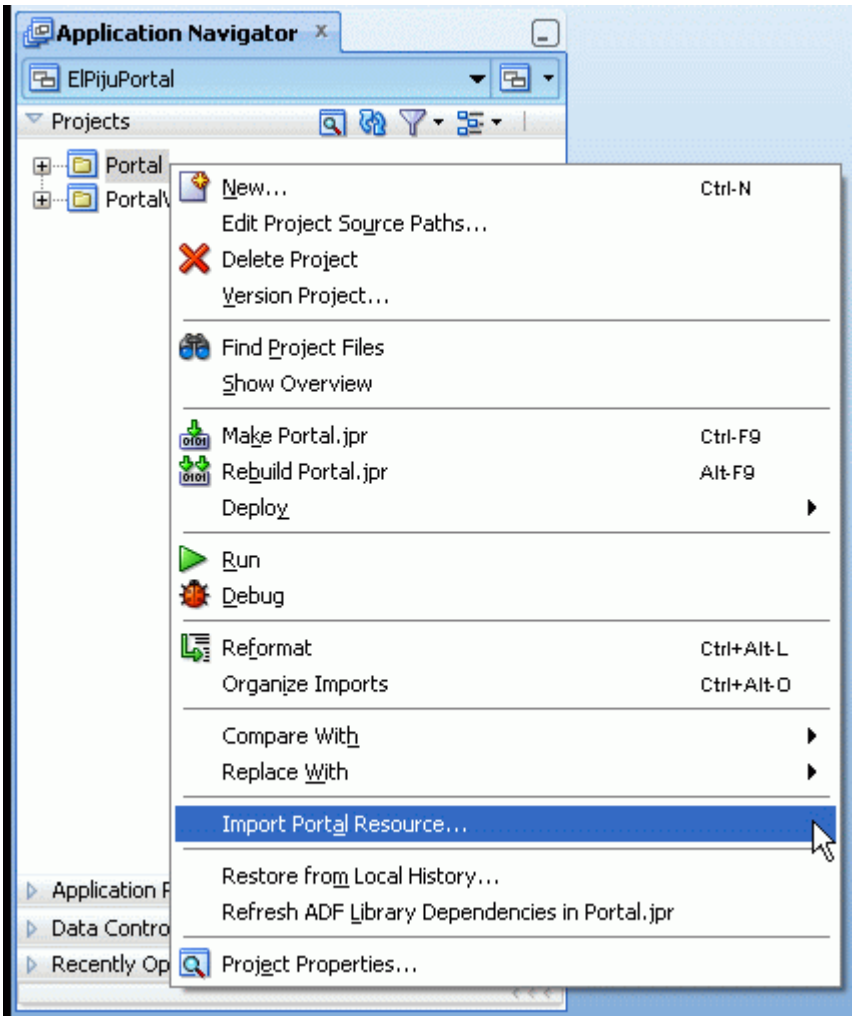


11. Import the portal resource.

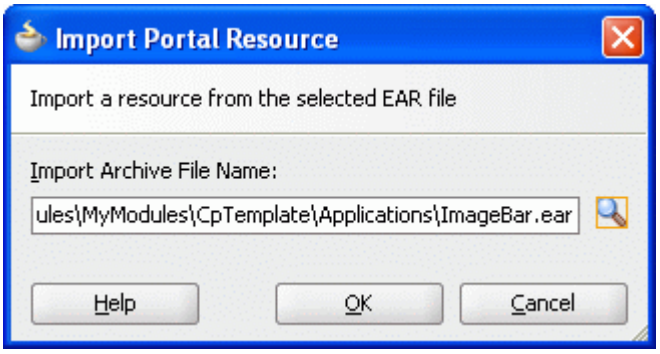
    Whether you developed it yourself or you received it from another developer, once you have a portal resource file, you can import it in JDeveloper into your custom WebCenter Portal application.

    Suppose you have created or opened a WebCenter Portal application in JDeveloper. It can be the El Piju Portal application, where we intend to use the newly developed page template.

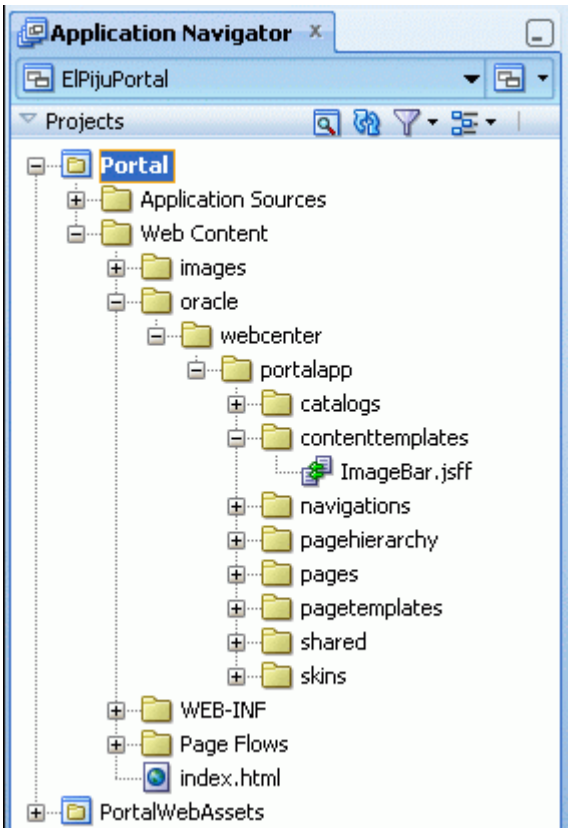
    a. In the Application Navigator, pane right-click the Portal project and choose **Import Portal Resource** from the pop-up menu.



b. In the Import Portal Resource window, provide the location of the portal resource archive. Click OK.



c. After importing you can expand the Web Content node and you will discover `ImageBar.jsff`. You can use the template in this application the same way as you did in step 8.



About Content Properties

In step 4, when you added the code to the body of the template, you accessed the actual item's information, that is, properties, using EL expressions.

Chapter 28.4.4 Using EL Expressions to Retrieve Content Item Information in the *Oracle® Fusion Middleware Developer's Guide for Oracle WebCenter* gives detailed information on the EL expressions and the available properties. Here we give only a brief overview and show a little example that displays the properties.

Each content item has a set of basic information that can be accessed by direct property names. Assuming that the actual item is stored in the `node` variable, here are a few examples:

`#{node.id}`	the item's unique identifier
`#{node.name}`	the item's name
`#{node.path}`	the item's path
`#{node.url.renderUrl}`	the URL to render the item in a browser
`#{node.isFolder}`	boolean value, true if the item is a folder

Each content item is associated with a specific content type defined in the Oracle Content Server repository. Content types can map to Content Server profile definitions and Site Studio Region Definitions. The content type defines the properties of the content item.

One of these properties is `primaryProperty`. Its value differs if the item is a document (`idcPrimaryFile`) or a folder (`dCollectionName`).

`#{node.primaryProperty}`	the node's primary property

All of these properties can also be accessed from the item:

`#{node.propertyMap}`	returns the properties as a Java Map object
`#{node.properties}`	returns the properties as an array

If you know the name of the property, you can access it using the property map.

`#{node.propertyMap['myProp']}`	The property called `myProp`

These properties are objects; you can access the object's components or test its type. In the next example, *`prop`* stands for `node.propertyMap['myProp'].` Some examples:

`#{`*`prop`*`.name}`	the name of the property
`#{`*`prop`*`.value}`	the value of the property
`#{`*`prop`*`.asTextHtml}`	the value of the property converted to a string
`#{`*`prop`*`.hasValue}`	test if the property has a value
`#{`*`prop`*`.type}`	the type of the property Note: It returns as an internal code, defined by the `oracle.webcenter.content.integration.Property` class. Not very useful, use the tests below.
`#{`*`prop`*`.isImage}`	test if the property is an image (JPEG, GIF, PNG, ...)
`#{`*`prop`*`.isTextBased}`	test if the property is text-based (`isHTML`, `isPlainText`, `isString`, `isCalendar`, `isBoolean`, `isDouble`, `isLong`)
`#{`*`prop`*`.isLink}`	test if the property is a link
`#{`*`prop`*`.isHTML}`	test if the property is in HTML format
`#{`*`prop`*`.isMsWorld}`	test if the property is in Microsoft Word format

Finally you can get the property's value in simple formats.

`#{`*`prop`*`.value.booleanValue}`	the value of the property as `java.lang.Boolean`
`#{`*`prop`*`.value.calendarValue}`	the value of the property as `java.util.Calendar`
`#{`*`prop`*`.value.stringValue}`	the value of the property as `java.lang.String`

The documentation also mentions how to discover the names of the properties for a given content type. In addition to these methods, you could use the small Content Presenter template, listed below. The template will list most of the basic information for any item, and then it – the code listed in bold – will display all of the item's properties and their value in a text format.

You can also download this template as a resource archive. ...

```
<?xml version='1.0' encoding='UTF-8'?>
<jsp:root xmlns:jsp="http://java.sun.com/JSP/Page" version="2.1"
          xmlns:dt="http://xmlns.oracle.com/webcenter/content/templates"
          xmlns:af="http://xmlns.oracle.com/adf/faces/rich"
          xmlns:f="http://java.sun.com/jsf/core"
          xmlns:c="http://java.sun.com/jsp/jstl/core"
          xmlns:fn="http://java.sun.com/jsp/jstl/functions">
  <dt:contentTemplateDef var="node">
    <af:panelFormLayout id="pfl1" rendered="#{not empty node}">
      <af:group id="grp1">
        <af:panelLabelAndMessage label="Basic Content Information"
                                 id="plam0"
                                 labelStyle="font-weight:bold;"/>
        <af:panelLabelAndMessage label="id" id="plam1">
          <af:outputText value="#{node.id}" id="ot1"/>
        </af:panelLabelAndMessage>
        <af:panelLabelAndMessage label="name" id="plam2">
          <af:outputText value="#{node.name}" id="ot2"/>
        </af:panelLabelAndMessage>
        <af:panelLabelAndMessage label="path" id="plam3">
          <af:outputText value="#{node.path}" id="ot3"/>
        </af:panelLabelAndMessage>
        <af:panelLabelAndMessage label="url.renderUrl" id="plam4">
          <af:outputText value="#{node.url.renderUrl}" id="ot4"/>
        </af:panelLabelAndMessage>
        <af:panelLabelAndMessage label="url.downloadUrl" id="plam5">
          <af:outputText value="#{node.url.downloadUrl}" id="ot5"/>
        </af:panelLabelAndMessage>
        <af:panelLabelAndMessage label="primaryProperty" id="plam6">
          <af:outputText value="#{node.primaryProperty}" id="ot6"/>
        </af:panelLabelAndMessage>
        <af:panelLabelAndMessage label="hasParentNode" id="plam7">
          <af:outputText value="#{node.hasParentNode}" id="ot7">
            <f:converter converterId="javax.faces.Boolean"/>
          </af:outputText>
        </af:panelLabelAndMessage>
        <af:panelLabelAndMessage label="parentId" id="plam8">
          <af:outputText value="#{node.parentId}" id="ot8"/>
        </af:panelLabelAndMessage>
        <af:panelLabelAndMessage label="createdBy" id="plam9">
          <af:outputText value="#{node.createdBy}" id="ot9"/>
        </af:panelLabelAndMessage>
        <af:panelLabelAndMessage label="createdDate" id="plam10">
          <af:outputText value="#{node.createdDate}" id="ot10"/>
        </af:panelLabelAndMessage>
        <af:panelLabelAndMessage label="modifiedBy" id="plam11">
          <af:outputText value="#{node.modifiedBy}" id="ot11"/>
        </af:panelLabelAndMessage>
        <af:panelLabelAndMessage label="modifiedDate" id="plam12">
          <af:outputText value="#{node.modifiedDate}" id="ot12"/>
        </af:panelLabelAndMessage>
        <af:panelLabelAndMessage label="folder/document?" id="plam21">
          <af:outputText value="#{node.isFolder ? 'folder': 'document'}"
                         id="ot21"/>
```

```
        </af:panelLabelAndMessage>
      </af:group>

      <af:group id="grp2">
        <af:panelLabelAndMessage label="Content Item Properties"
                                 id="plam30"
                                 labelStyle="font-weight:bold;">
          <af:outputText value="#{fn:length(node.propertyMap)}"
                         id="ot30"/>
        </af:panelLabelAndMessage>
        <af:forEach items="#{node.properties}" var="prop">
          <af:panelLabelAndMessage label="#{prop.name}" id="plam31">
            <af:outputText value="#{prop.asTextHtml}" id="ot45"/>
          </af:panelLabelAndMessage>
        </af:forEach>
      </af:group>
    </af:panelFormLayout>
  </dt:contentTemplateDef>
</jsp:root>
```

On the following a screen shot you can see the template's output used with an image file. The screen shot is not complete; there are more properties below the bottom of the image.

Basic Content Information	
id	/UCM/DADVMC0302U5OR001402
name	kacaj.JPG
path	/UCM/Contribution Folders/images/kacaj.JPG
url.renderUrl	http://127.0.0.1:7101/CPTemplates/ShowProperty?nodeId=%2FUCM%2FDADVMC0302U5OR001402%2F%2FidcPrimaryFile&revision=latestreleased
url.downloadUrl	http://127.0.0.1:7101/CPTemplates/DownloadBinaryServlet?nodeId=%2FUCM%2FDADVMC0302U5OR001402&propertyId=%2FUCM%2FDADVMC0302U5OR001402%2Fprimary&fileName=kacaj.JPG
primaryProperty	idcPrimaryFile: kacaj.JPG
hasParentNode	true
parentId	/UCM/IDC:Folder/703354910487002201
createdBy	weblogic
createdDate	11/22/2011
modifiedBy	weblogic
modifiedDate	11/22/2011
folder/document?	document
Content Item Properties	47
idcPrimaryFile	
idcRenditions	
dID	1402
dDocName	DADVMC0302U5OR001402
dDocType	Document
dDocTitle	kacaj.JPG
dDocAuthor	weblogic
dRevClassID	1402
dRevisionID	1
dRevLabel	1
dRevRank	0
dIsCheckedOut	0
dCheckoutUser	null
dSecurityGroup	Public
dDocAccount	null
dCreateDate	Tue Nov 22 16:48:00 CET 2011
dInDate	Tue Nov 22 16:48:00 CET 2011
dOutDate	null
dStatus	RELEASED
dReleaseState	Y
dReleaseDate	Sat Jan 07 18:08:46 CET 2012
dWorkflowState	null
dFormat	image/jpeg
VaultFileSize	1823725
dWebExtension	jpg
xComments	The funniest, cutest laugh!
xClbraUserList	null
xClbraAliasList	null
xClbraRoleList	null

Create Image Gallery Viewer Template

In this example, you will create a more sophisticated image viewer template. The template will display thumbnail images from a collection.

For this example I would like to thank John Brunswick, who published the original template in his podcast at http://www.johnbrunswick.com/2011/06/e2-0-workbench-podcast-5-%E2%80%93-webcenter-image-gallery-with-content-presenter/. I modified his code to adapt to the changes in WebCenter Portal and WebCenter Content 11g Release 1 (11.1.1.6.0), as well as the changes in the

fancyBox 2.0 framework. The template also uses the `shared` folder to pack all the required files into a portal resource archive.

At the bottom of the page you can see a set of thumbnail images of a construction site. The new template displays this thumbnail gallery.

If you click on any of the thumbnails, the original image will open in a window, resized to fit the browser. This pop-up window can also display image's comments and you can browse to the previous/next images.

For the pop-up image display you will use *fancyBox 2*, a jQuery based framework, that can be downloaded from http://fancyapps.com/fancybox/. Note: We use this framework as an illustration of how to combine WebCenter Portal applications with custom JavaScript. If you want to use the framework in a real application, consult the Web site and contact the author for copyright and licensing issues. Consult the documentation on the Web site if you want to use extra features of the framework not used in our example.

Before you start using *fancyBox*, here is a quick overview about how to use the framework:

- Load the required JavaScript and CSS libraries.

```
<script type="text/javascript"
  src="http://ajax.googleapis.com/ajax/libs/jquery/1.7/jquery.min.js" />
<link rel="stylesheet"
   href="/fancybox/source/jquery.fancybox.css?v=2.0.4" type="text/css"
   media="screen" />
<script type="text/javascript"
   src="/fancybox/source/jquery.fancybox.pack.js?v=2.0.4" />
```

  Note: We are not using some of the optional libraries.

- Initialize fancyBox

```
<script type="text/javascript">
   $(document).ready(function() {
         $(".fancybox").fancybox();
   });
</script>
```

This code associates the fancyBox processing with a CSS style name; in this example it is called: `fancybox`. You can also pass initialization parameters here.

- Create the image gallery

```
<a class="fancybox" rel="group" href="big_image_1.jpg" title="First">
  <img src="small_image_1.jpg" alt="" />
</a>
<a class="fancybox" rel="group" href="big_image_2.jpg" title="Second">
  <img src="small_image_2.jpg" alt="" />
</a>
```

It is fairly simple. Let's see how to translate it to ADF Faces and apply it in a Content Presenter template.

12. Create a new Content Presenter template.

    Follow the process outlined in steps 2 and 3, create a new template file: `ImageGallery.jsf`. This template will also process multiple items.

    Here is the template source so far:

```
<?xml version='1.0' encoding='UTF-8'?>
<jsp:root
     xmlns:jsp="http://java.sun.com/JSP/Page" version="2.1"
     xmlns:dt="http://xmlns.oracle.com/webcenter/content/templates">
  <dt:contentListTemplateDef var="nodes"/>
</jsp:root>
```

13. Populate the template body.

    a. Download the required JavaScript and CSS libraries.

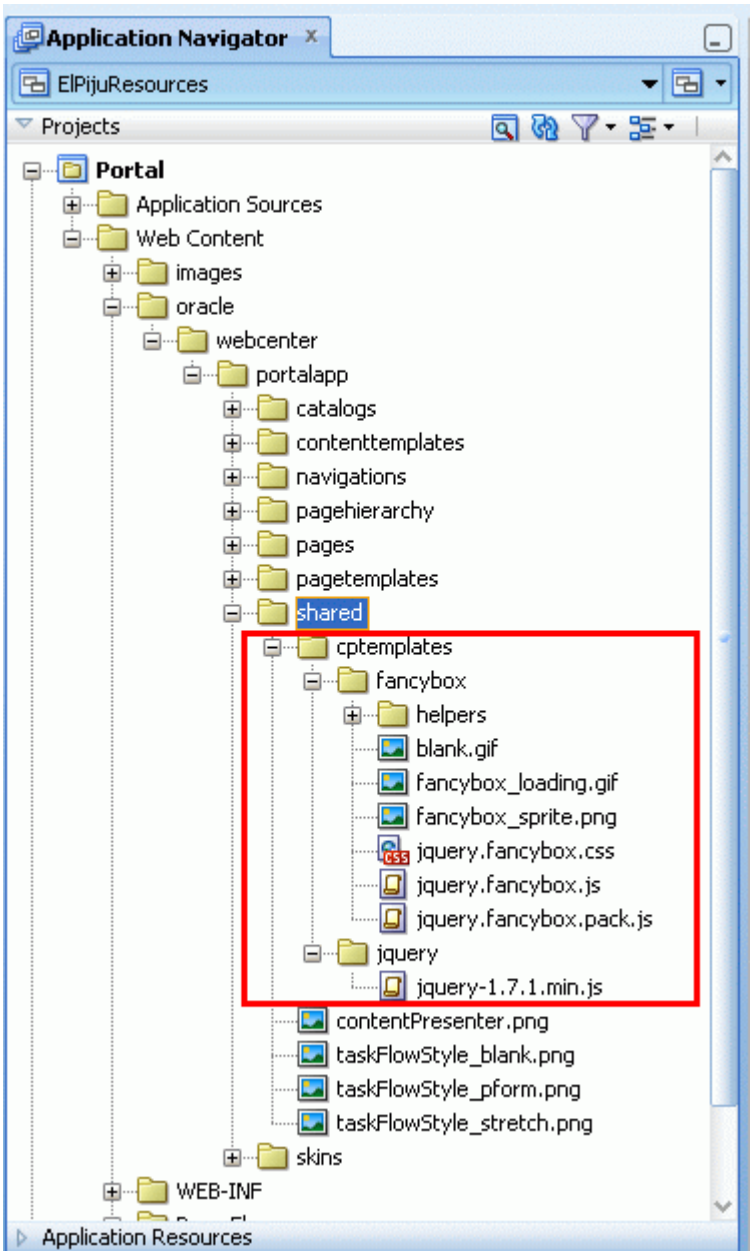
       As you can see above, you could access the jQuery library directly from the site `ajax.googleapis.com`, but it requires that your browser accesses another site. Besides, the fancyBox site does not provide a direct URL to use their scripts, so you have to download them.

       Download the fancyBox distribution ZIP file, save it in your system, and unzip it. You will need the content of the `source` folder. Also download and save the jQuery library from http://ajax.googleapis.com/ajax/libs/jquery/1.7/jquery.min.js.

       There is a special folder in WebCenter Portal applications: `/oracle/webcenter/portalapp/shared`.

       The contents of this folder and its subfolders can be packed into a resource archive so that you can dynamically upload them into an application. Note: Although this folder is present since WebCenter Portal 11g Release1 (11.1.1.4.0), accessing non-image files and files from subfolders is only available since version 11.1.1.6.0. So the following trick requires that you run at least version 11.1.1.6.0.

Locate the application's root folder, and then find the shared folder on your disk. We recommend that you create a subfolder: **`cptemplates`** in the shared folder, then copy the jQuery library to a subfolder called **`jquery`**, and the source folder of the fancyBox framework to a subfolder called `fancybox`. Once you are done copying the files, select **View > Refresh** from the menu to refresh the portal project. This is how the shared folder will look:



b. Add two more namespaces – indicated in bold - the root tag. We will use them later:

```
<jsp:root xmlns:jsp="http://java.sun.com/JSP/Page" version="2.1"
          xmlns:dt="http://xmlns.oracle.com/webcenter/content/templates"
          xmlns:af="http://xmlns.oracle.com/adf/faces/rich"
          xmlns:c="http://java.sun.com/jsp/jstl/core">
```

c. Load the required JavaScript and CSS code.

ADF Faces `<af:resource>` tag lets you load JavaScript and CSS files into your page.

Add the following three `<af:resources>` tags, noted in bold, to your page and place them right after the `<jsp:root>` element. Please note that the `source` attribute contains relative links to the files placed under the `shared` folder.

```
<?xml version='1.0' encoding='UTF-8'?>
<jsp:root xmlns:jsp="http://java.sun.com/JSP/Page" version="2.1"
          xmlns:dt="http://xmlns.oracle.com/webcenter/content/templates"
          xmlns:af="http://xmlns.oracle.com/adf/faces/rich"
          xmlns:c="http://java.sun.com/jsp/jstl/core">
  <af:resource type="javascript"
   source="/oracle/webcenter/portalapp/shared/cptemplates/jquery/jquery-
1.7.1.min.js"/>
  <af:resource type="javascript"
source="/oracle/webcenter/portalapp/shared/cptemplates/fancybox/jquery.f
ancybox.js"/>
  <af:resource type="css"
source="/oracle/webcenter/portalapp/shared/cptemplates/fancybox/jquery.f
ancybox.css"/>
```

d. Initialize the fancyBox code.

The `<af:resources>` tag enables you to place JavaScript code directly into the page. Add the following code:

```
  <af:resource type="javascript">
$(document).ready(function()
{
  $('a.grouped_elements').fancybox
  (
    {
      transitionIn  : 'none',
      transitionOut : 'none',
      loop          : false,
      openEffect    : 'none',
      closeEffect   : 'none',
      nextEffect    : 'none',
      prevEffect    : 'none',
      overlayShow   : false,
      helpers       : { title: { type: 'inside' } },
      afterLoad     : function()
        { this.title = 'Image ' +
                       (this.index + 1) + ' / ' + this.group.length +
                       (this.title ? ' - ' + this.title : '');}
    }
  );
});
  </af:resource>
```

This code will associate fancyBox with a style called `grouped_elements` and sets some options, including how the title is created. The title will be formatted like

Image *n* / *m* – *original title*

e. Add local style definition.

As you can see in the screen shot, the thumbnail images have a narrow green border. To create such a border, add the following CSS style definition to the template. The `<af:resources>` tag can be used again.

```
  <af:resource type="css">
img.gallery
{
  border: 1px solid green;
  padding: 2px;
  margin: 10px 10px 10px 0;
  vertical-align: top;
}
  </af:resource>
```

f. Add an iterator inside the template definition tag.

This template will be called with multiple items, so you need an iterator to process all of the items. The code that you have to add inside the `<dt:contentListTemplateDef>` element is in bold below:

```
  <dt:contentListTemplateDef var="nodes">
    <af:iterator rows="0" var="node" value="#{nodes}" id="it0">
    </af:iterator>
  </dt:contentListTemplateDef>
```

g. Now you have to produce the `<a>` and embedded `<img>` tags that will be recognized and processed by fancyBox. You want to create the following HTML fragment:

```
01 <a class="grouped_elements fancybox.image" rel="group"
02    href="#{node.primaryProperty.renderUrl}"
03    title="image title">
04   <img src="thumbnail url" class="gallery" />
05 </a>
```

Line 01

In this code, the **`grouped_element`** class is the one which is associated in the initialization code with the fancyBox processing.

The other class, **`fancybox.image`**, is a marker class defined in the fancyBox CSS code. Using this class tells fancyBox that it has to display an image. Without such a class definition fancyBox will try to guess what kind of file would be rendered from the file's extension or MIME type. This guessing proved to be unreliable with Content Server, so we added the marker class name.

Line 02

You have already used the EL expression: `#{node.primaryProperty.renderUrl}`, which refers to the URL of the actual – full sized – image.

Line 03

Each document in the content server might have a comment attribute. We will display this attribute as the image's title, or use the file name if the comment attribute is empty. The following EL expression will display the comment or the file name.

```
title="#{node.propertyMap['xComments'].hasValue ?
           node.propertyMap['xComments'].asTextHtml :
           node.propertyMap['dDocTitle'].value.stringValue}"
```

Line 04

The `class=gallery` attribute will produce the green border around the thumbnail image.

The biggest problem is to get a thumbnail image for all of the images. For the time being, let's cheat and use the original image, resized by the browser. Use this code in line 4.

```
  <img src="#{node.primaryProperty.renderUrl}" class="gallery"
       width="100"/>
```

In general it is a bad compromise, but we will show you a better solution in the next step.

h. So how do you enter the above pure HTML fragment into a JSFF page? Although technically you can mix JSF (ADF Faces) tags with pure HTML tags, we found it a better practice to leverage ADF Faces tags like af:outputText. It seems that ADF Faces often rearranges and changes the pure HTML tags before the actual page is rendered.

The `<img>` tag is simple, it can easily be replaced with an `<af:image>` tag. The `<a>` tag is more complicated. A reliable solution is to inject the actual tag using `<af:outputText escape="false" value="...">` but of course the HTML tag has to be escaped.

Enter the following code inside the `<af:iterator>` element.

```
<af:outputText escape="false"
    value="&lt;a class=&quot;grouped_elements fancybox.image&quot;
rel=&quot;group&quot; href=&quot;#{node.url.renderUrl}&quot;
title=&quot;#{node.propertyMap['xComments'].hasValue ?
node.propertyMap['xComments'].asTextHtml :
node.propertyMap['dDocTitle'].value.stringValue}&quot;&gt;"/>
  <af:image source="#{node.url.renderUrl}" styleClass="gallery"
      inlineStyle="width:100.0px;"/>
<af:outputText escape="false" value="&lt;/a&gt;"/>
```

14. Use Content Presenter's thumbnail images.

The above workaround to create thumbnail images letting the browser resize the full image is often inadequate; in the case of large images, it takes too long to render the thumbnail gallery.

Fortunately Content Server can automatically create thumbnail images. When Oracle WebCenter Content Server is installed with Inbound Refinery component and it is properly configured, Inbound Refinery will automatically create a thumbnail image for all the images uploaded to the Content Server. For the details of configuring Content Server for WebCenter Portal, see our training module: *Installing and Configuring Enterprise Content Management for Oracle WebCenter Portal: Spaces.*

There is a little bit more work to do. In the above example, when we used `src="#{node.primaryProperty.renderUrl}"`, it translated to an URL similar to:

`src="http://<server host>:<server port>/<context root>/ShowProperty?nodeId=/ElPijuDocuments/DADVMC0302USOR001836//idcPrimaryFile&revision=latestreleased`

It means that the application contains a special servlet, `ShowProperty`, which contacts the Content Server using the VCR adapter and requests the rendering of the latest released version of the document with the given `nodeId`. Currently - in 11.1.1.6.0 - the servlet can access only the default - normal - not the thumbnail rendering. The thumbnail rendering is not directly accessible from a custom WebCenter Portal application.

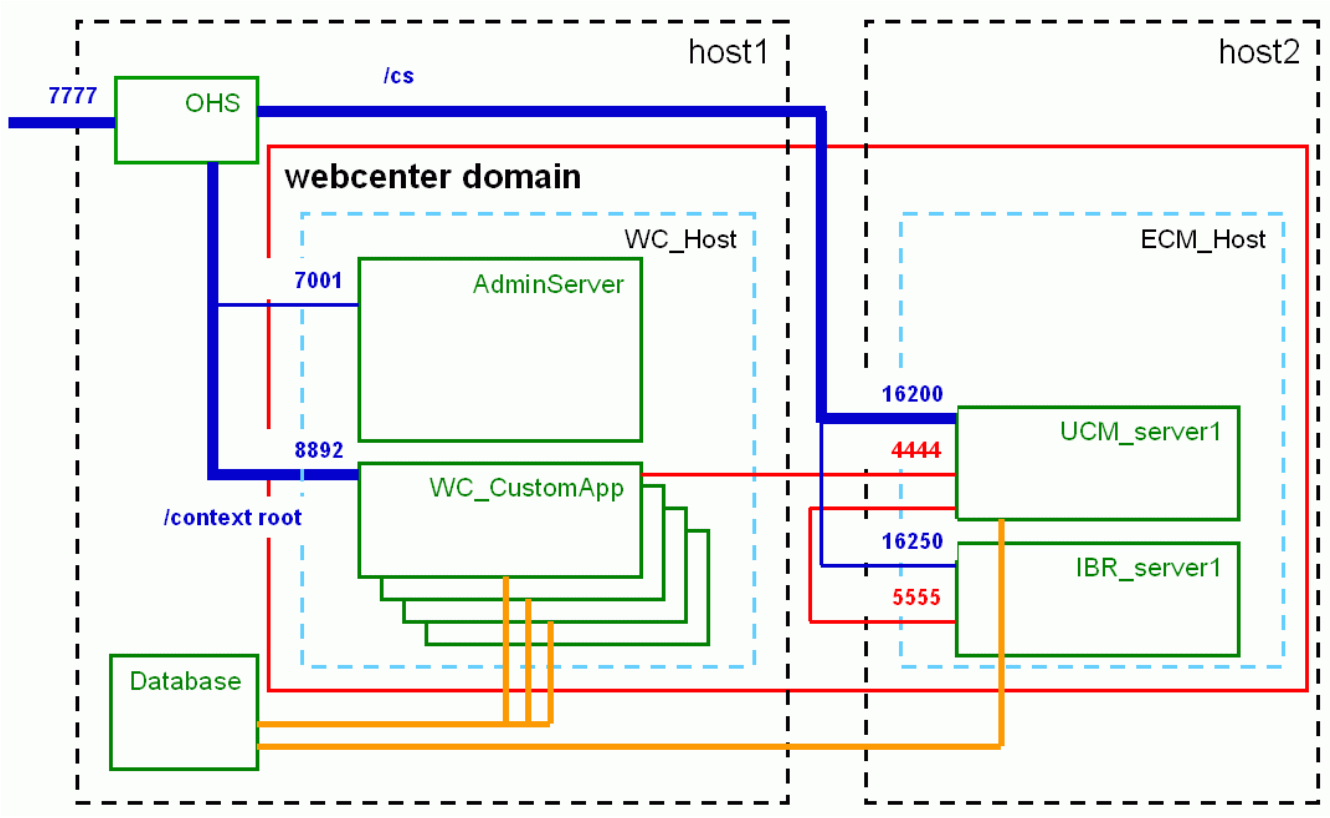
The thumbnail rendering is accessible only via Content Server's Web interface.

```
src="http://<content server host>:<content server Web port>/cs/idcplg?IdcService=GET_FILE&dID=2036&Rendition=rendition:T"
```

Although it is easy to obtain the `dID` attribute - 2036 in the above example - value using `#{node.propertyMap['dID'].value}`, it is not possible to programmatically get the first part of the URL, which is printed in bold above: the host , the port, and the URL prefix where the Content Server Web interface can be accessed. These values depend on the actual topology, that is, how the Content Server was installed, and where our portal application is deployed.

In a typical, recommended installation, an Oracle HTTP server is used as a reverse proxy to access both the custom application's context root and Content Server's Web interface. This means that the `<server host>` and `<content server host>`, and `<server port>` and `<content server port>` will be the same value, that of `<OHS server host>` and `<OHS server port>`. In practical terms, you could use a simple URL, without specifying host name and port number, such as:

```
src="/cs/idcplg?IdcService=GET_FILE&dID=2036&Rendition=rendition:T"
```



However we do not want to rely on this topology. This topology is definitely not true during development when your application runs in JDeveloper only.

a. To solve this problem, we introduce a variable that will contain the first part of the Content Server access URL, i.e.

http://*<content server host>*:*<content server Web port>*/cs

Note: This URL is system dependent, so you might have to change it before the template is exported as a portal resource, or before the template is used. You will see in the next step that you can do this modification at run time as well.

Add the following variable definition just before the `<dt:contentListTemplateDef>` tag.

```
<c:set var="url"
       value="http://<content server host>:<content server Web port>/cs"
       scope="session" />
```

b. Change the `<af:image>` tag to use the URL that accesses the thumbnail rendering. Note that the `source` attribute refers to the above-defined `url` variable. Use this code:

```
<af:image source=
"#{url}/idcplg?IdcService=GET_FILE&amp;dID=#{node.propertyMap['dID'].val
ue}&amp;Rendition=rendition:T"
          styleClass="gallery"/>
```

Finally here is the complete source of the page template:

```
<?xml version='1.0' encoding='UTF-8'?>
<jsp:root xmlns:jsp="http://java.sun.com/JSP/Page" version="2.1"
          xmlns:dt="http://xmlns.oracle.com/webcenter/content/templates"
          xmlns:af="http://xmlns.oracle.com/adf/faces/rich"
          xmlns:c="http://java.sun.com/jsp/jstl/core">
  <af:resource type="javascript"
source="/oracle/webcenter/portalapp/shared/cptemplates/jquery/jquery-
1.7.1.min.js"/>
  <af:resource type="javascript"
source="/oracle/webcenter/portalapp/shared/cptemplates/fancybox/jquery.f
ancybox.js"/>
  <af:resource type="css"
source="/oracle/webcenter/portalapp/shared/cptemplates/fancybox/jquery.f
ancybox.css"/>

  <af:resource type="javascript">
$(document).ready(function()
{
  $('a.grouped_elements').fancybox
  (
    {
      transitionIn  : 'none',
      transitionOut : 'none',
      loop          : false,
      openEffect    : 'none',
      closeEffect   : 'none',
      nextEffect    : 'none',
      prevEffect    : 'none',
      overlayShow   : false,
      helpers       : { title: { type: 'inside' } },
```

```
      afterLoad     : function()
        { this.title = 'Image ' +
                       (this.index + 1) + ' / ' + this.group.length +
                       (this.title ? ' - ' + this.title : '');}
    }
  );
});
  </af:resource>

  <af:resource type="css">
img.gallery
{
  border: 1px solid green;
  padding: 2px;
  margin: 10px 10px 10px 0;
  vertical-align: top;
}
  </af:resource>

<!-- Edit this variable to point to the Content Server's Web UI -->
  <c:set var="url"
         value="http://dadvmh0213.us.oracle.com:7777/cs"
         scope="session" />

  <dt:contentListTemplateDef var="nodes">
    <af:iterator rows="0" var="node" value="#{nodes}" id="it0">
      <af:outputText escape="false"
          value="&lt;a class=&quot;grouped_elements fancybox.image&quot;
rel=&quot;group&quot; href=&quot;#{node.url.renderUrl}&quot;
title=&quot;#{node.propertyMap['xComments'].hasValue ?
node.propertyMap['xComments'].asTextHtml :
node.propertyMap['dDocTitle'].value.stringValue}&quot;&gt;"/>
        <af:image
source="#{url}/idcplg?IdcService=GET_FILE&amp;dID=#{node.propertyMap['dI
D'].value}&amp;Rendition=rendition:T"
          styleClass="gallery"/>
      <af:outputText escape="false" value="&lt;/a&gt;"/>
      </af:iterator>
  </dt:contentListTemplateDef>
</jsp:root>
```

15. Create the portal resource and then test the template.

    a. As described in step 6, create a portal resource out of the `ImageGallery.jsff` file. Set the Display Name to **`Image Gallery`**, and set the view ID to **`elpiju.content.templates.imagegallery`**. Do not forget to change the visibility attribute to **`TRUE`**.

    b. If you want, you can test the template within the `ElPijuResources` application, as described in step 7.

       If you use our content uploaded to the content server, you will find some folders with images for El Piju's reference construction sites. For example, you can drop the folder

as a Content Presenter task flow and use the view ID `elpiju.content.templates.imagegallery`.

However in the next step we will show you how you can test the template at run time.

Use the Content Presenter Template at Run Time

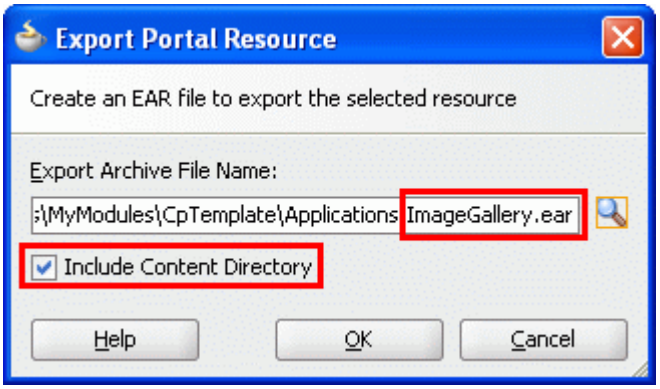
In this part of the training module you will see how to work with a Content Presenter template at run time. First you will import the portal resource containing the template into the running application. Later you will add a Content Presenter task flow and configure it to use the new template.

We are going to use the El Piju application. We assume that you have downloaded it and run it in JDeveloper or it is deployed to an Oracle WebCenter Portal instance that you can access.

If you do not have El Piju, you can still follow the steps using any WebCenter Portal application (even Spaces), as long as you have access to a Content Server with a set of images and their thumbnails.

16. Export the template as a portal resource.

a. While still in JDeveloper, follow the process outlined in step 10; create a portal resource archive with the name **`ImageGallery.ear`**. The important difference is that now you must select the **Include Content Directory** checkbox, since we have additional files stored in the `shared` folder.



b. If you open the `ImageGallery.ear` file, you will discover that the content of the shared folder and subfolders are now included in the `transport.mar` archive.

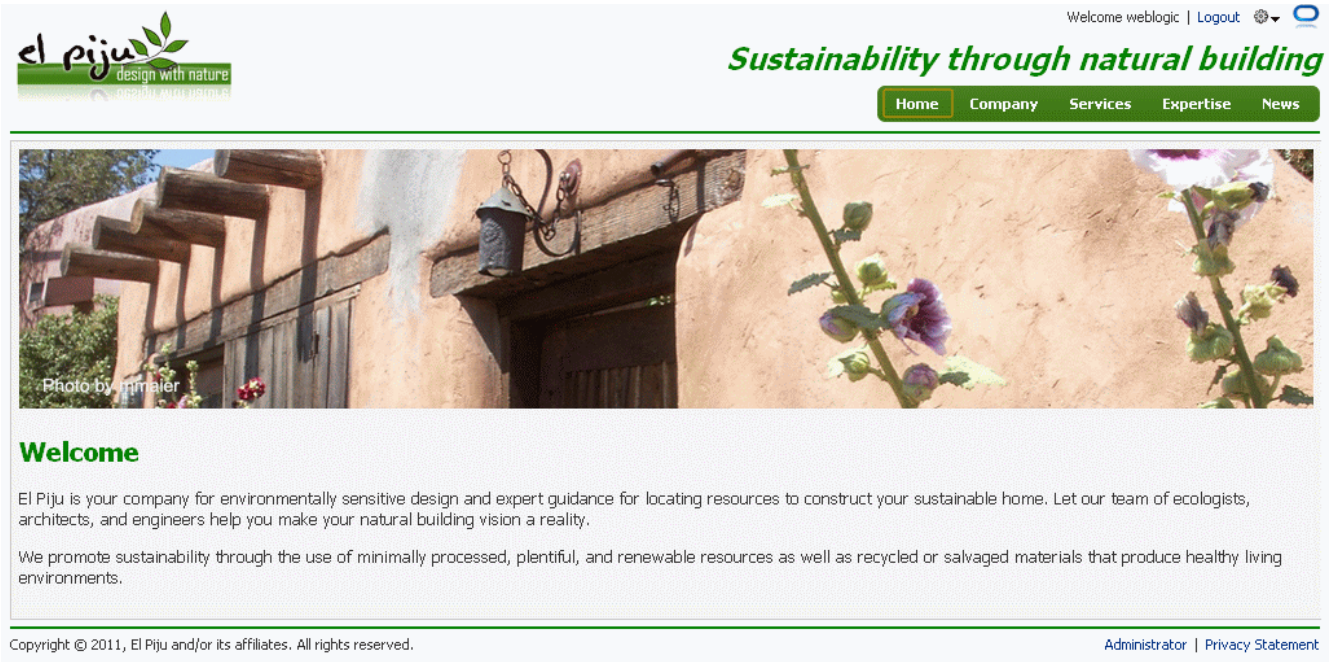
ImageGallery.ear | transport.mar

Path	Date	Size	Compressed
oracle/webcenter/portalapp/contenttemplates/ImageGallery.jsff	1/24/12 2:06 PM	2.69 KB	1.08 KB
oracle/webcenter/portalapp/shared/contentPresenter.png	1/24/12 2:06 PM	1.14 KB	1.15 KB
oracle/webcenter/portalapp/shared/cptemplates/fancybox/blank.gif	1/24/12 2:06 PM	43	39
oracle/webcenter/portalapp/shared/cptemplates/fancybox/fancybox_loading.gif	1/24/12 2:06 PM	5.83 KB	5.37 KB
oracle/webcenter/portalapp/shared/cptemplates/fancybox/fancybox_sprite.png	1/24/12 2:06 PM	2.69 KB	2.69 KB
oracle/webcenter/portalapp/shared/cptemplates/fancybox/helpers/fancybox_buttons.png	1/24/12 2:06 PM	2.34 KB	2.34 KB
oracle/webcenter/portalapp/shared/cptemplates/fancybox/helpers/jquery.fancybox-buttons.css	1/24/12 2:06 PM	1.77 KB	585
oracle/webcenter/portalapp/shared/cptemplates/fancybox/helpers/jquery.fancybox-buttons.js	1/24/12 2:06 PM	2.79 KB	1,017
oracle/webcenter/portalapp/shared/cptemplates/fancybox/helpers/jquery.fancybox-thumbs.css	1/24/12 2:06 PM	719	280
oracle/webcenter/portalapp/shared/cptemplates/fancybox/helpers/jquery.fancybox-thumbs.js	1/24/12 2:06 PM	3.57 KB	1.38 KB
oracle/webcenter/portalapp/shared/cptemplates/fancybox/jquery.fancybox.css	1/24/12 2:06 PM	3.59 KB	1.06 KB
oracle/webcenter/portalapp/shared/cptemplates/fancybox/jquery.fancybox.js	1/24/12 2:06 PM	29.18 KB	8.44 KB
oracle/webcenter/portalapp/shared/cptemplates/fancybox/jquery.fancybox.pack.js	1/24/12 2:06 PM	15.56 KB	5.71 KB
oracle/webcenter/portalapp/shared/cptemplates/jquery/jquery-1.7.1.min.js	1/24/12 2:06 PM	91.67 KB	32.39 KB
oracle/webcenter/portalapp/shared/taskFlowStyle_blank.png	1/24/12 2:06 PM	528	483
oracle/webcenter/portalapp/shared/taskFlowStyle_pform.png	1/24/12 2:06 PM	2.16 KB	2.16 KB
oracle/webcenter/portalapp/shared/taskFlowStyle_stretch.png	1/24/12 2:06 PM	1.77 KB	1.77 KB
oracle/webcenter/siteresources/scopedMD/s8bba98ff_4cbb_40b8_beee_296c916a23ed/generic-site-resources.xml	1/24/12 2:06 PM	1.85 KB	645

17. Import and edit the template if needed.

    a. Switch to a browser and access the running application. Log in as a user with administrative privileges.

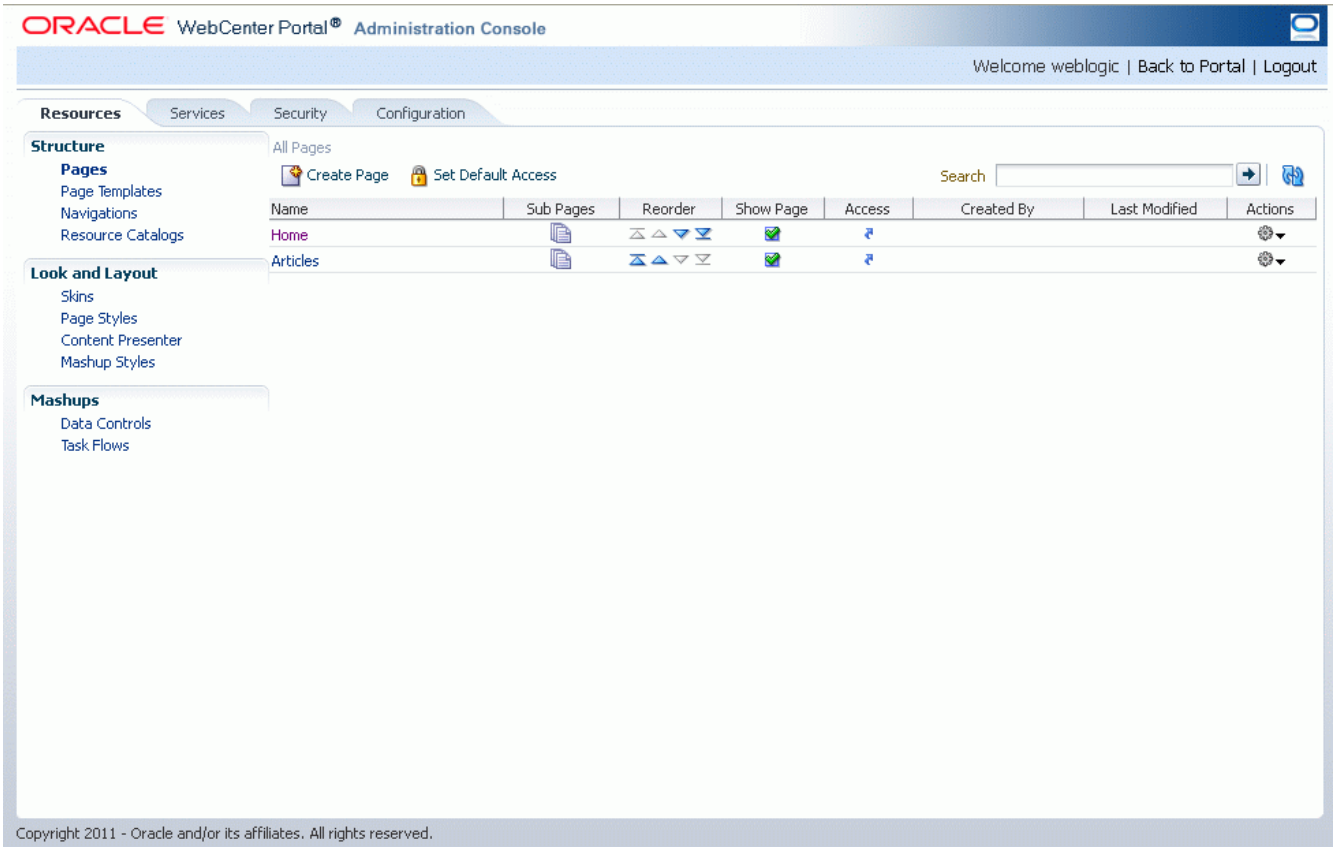
       Here is El Piju application's home page with `weblogic` user logged in.



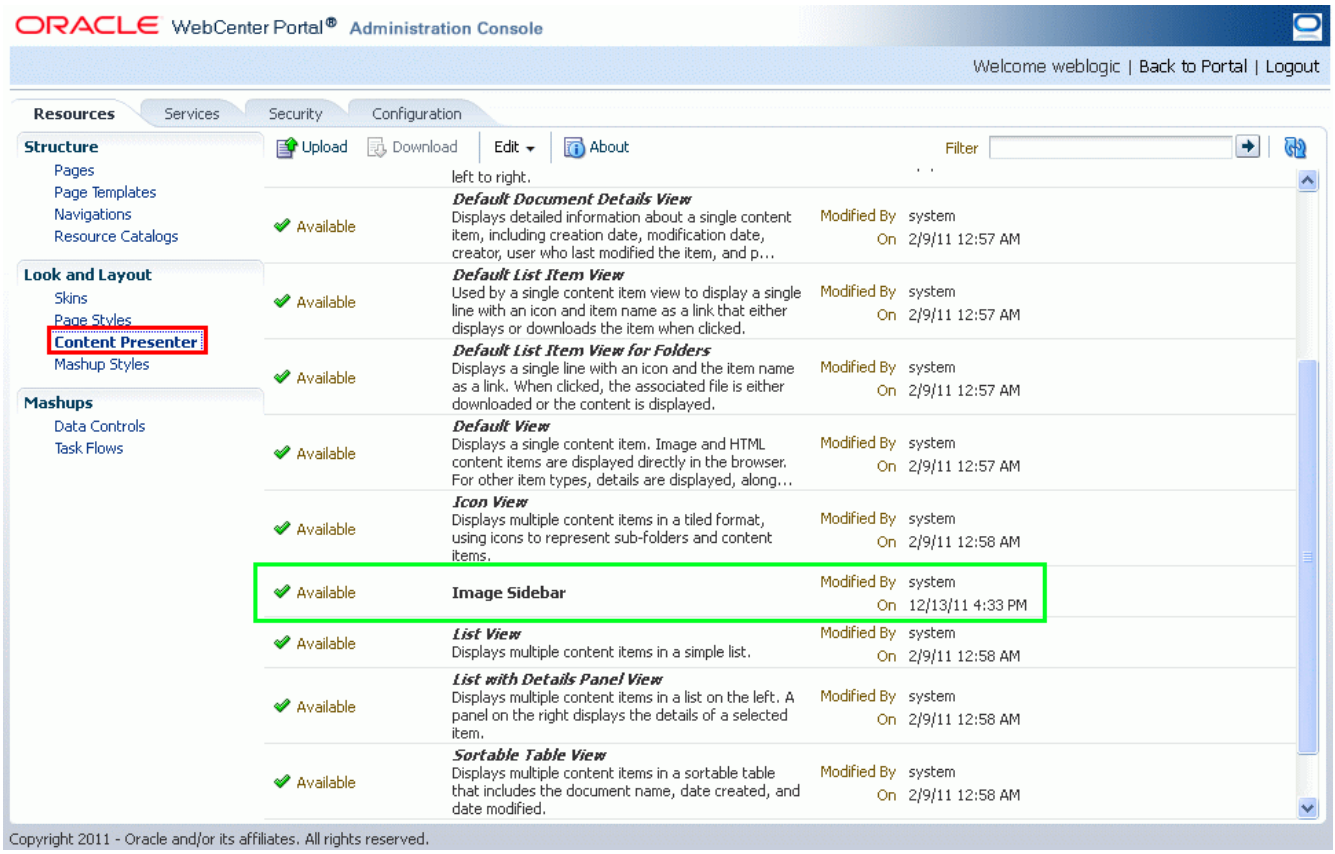
    b. Access the Administration Console.

       Depending on the application's navigation model, there might be various ways to access the Administration console. In El Piju there is a pop-up menu in the page header, as you see in the next screen shot.

If you don't find the navigation in your application, try the following URL: `http://<server>:<port>/<context root>/admin`. Keep in mind that this page is accessible only for users with administrator privileges.

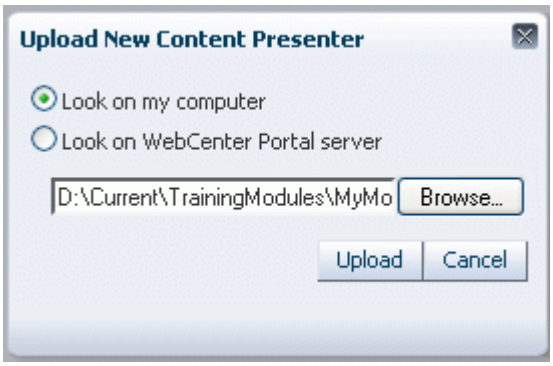


c. In the Resources tab click the **Content Presenter** link.

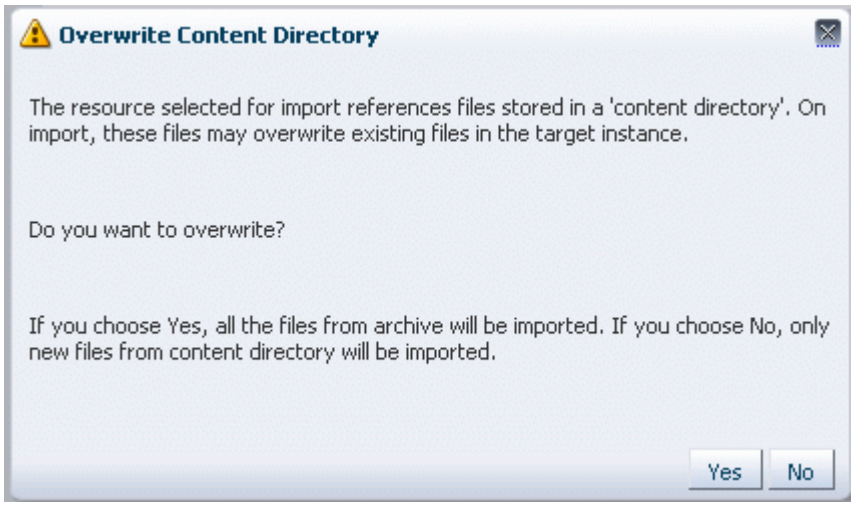


This page shows all the available Content Presenter templates. You can see the out-of-the-box templates with their name and description displayed in italics. On the screen shot you also see a custom template, Image Sidebar, the one you created earlier in this training. It is owned and modified by `system`. This indicates that the template was added to the application at design time.

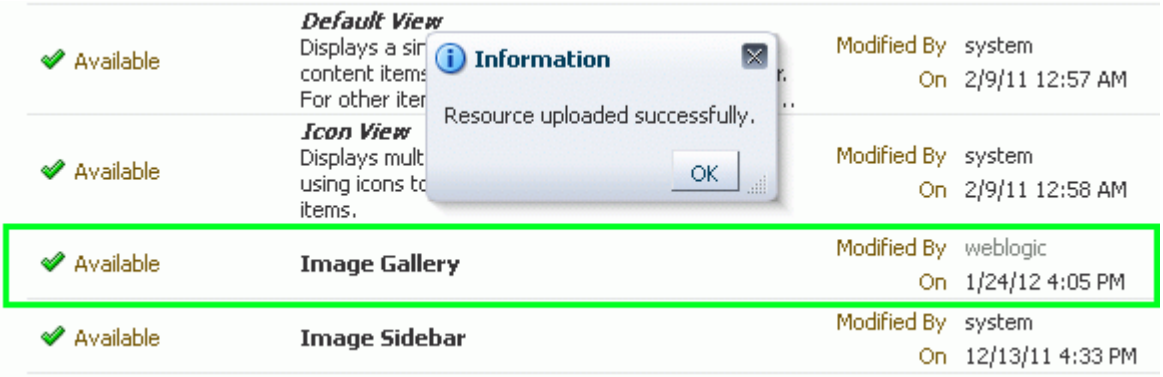
d. Click the Upload link at the top of the list. In the pop-up window, locate the portal resource archive **`ImageGallery.ear`** on your local system and click the **Upload** button.



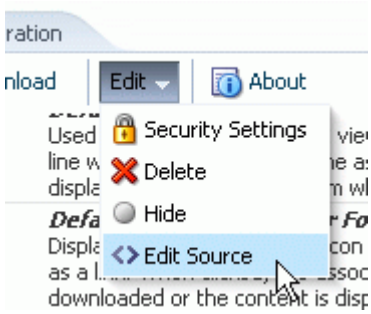
Since the archive contain also files of the `shared` folder, you will get a warning that files might get overwritten in the upload process. When you click **No**, only new files will be imported.



After the import finishes, a confirmation window displays. In this screenshot you can also see the newly imported template, Image Gallery. Note that it is modified by `weblogic`, the current user who uploaded the resource. Also note that the template is Available; this is the result of setting the `visible` attribute to true in the `generic-site-resources.xml` file.



e. When the Image Gallery template's line is selected, you can choose **Edit Source** from the Edit menu.



Here in the Edit Source window, you can modify the template source. For example, change the Content Server Web Access URL, we discussed in step 14a.

18. Use the Content Presenter template.

    You are going to edit a page, add the Content Presenter task flow, and configure the content and the template the task flow will use.

    a. Edit a page.

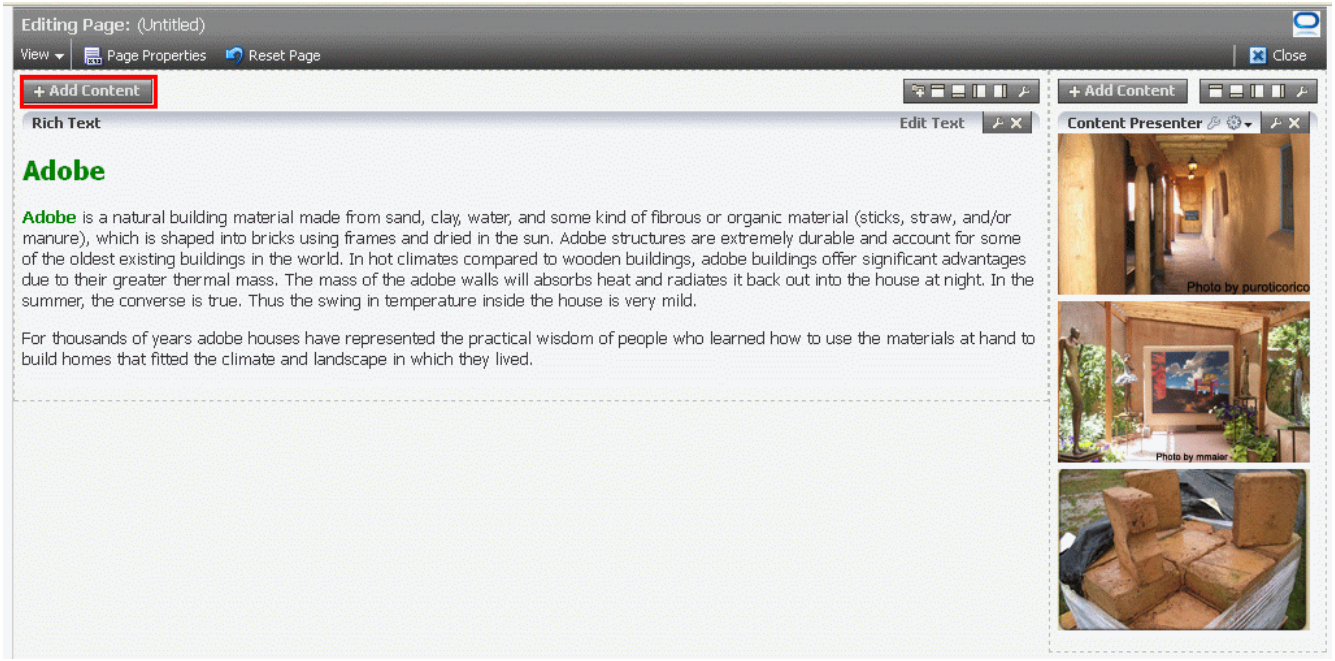
       In the El Piju application you will edit the Adobe page, but if you use another application, just pick any page.

       Depending on the application, there are various ways to switch to edit mode. Generally you can use the Administrator Console's Resources tab. Here in El Piju, if you have edit privilege, you will see a link in the pop-up menu.

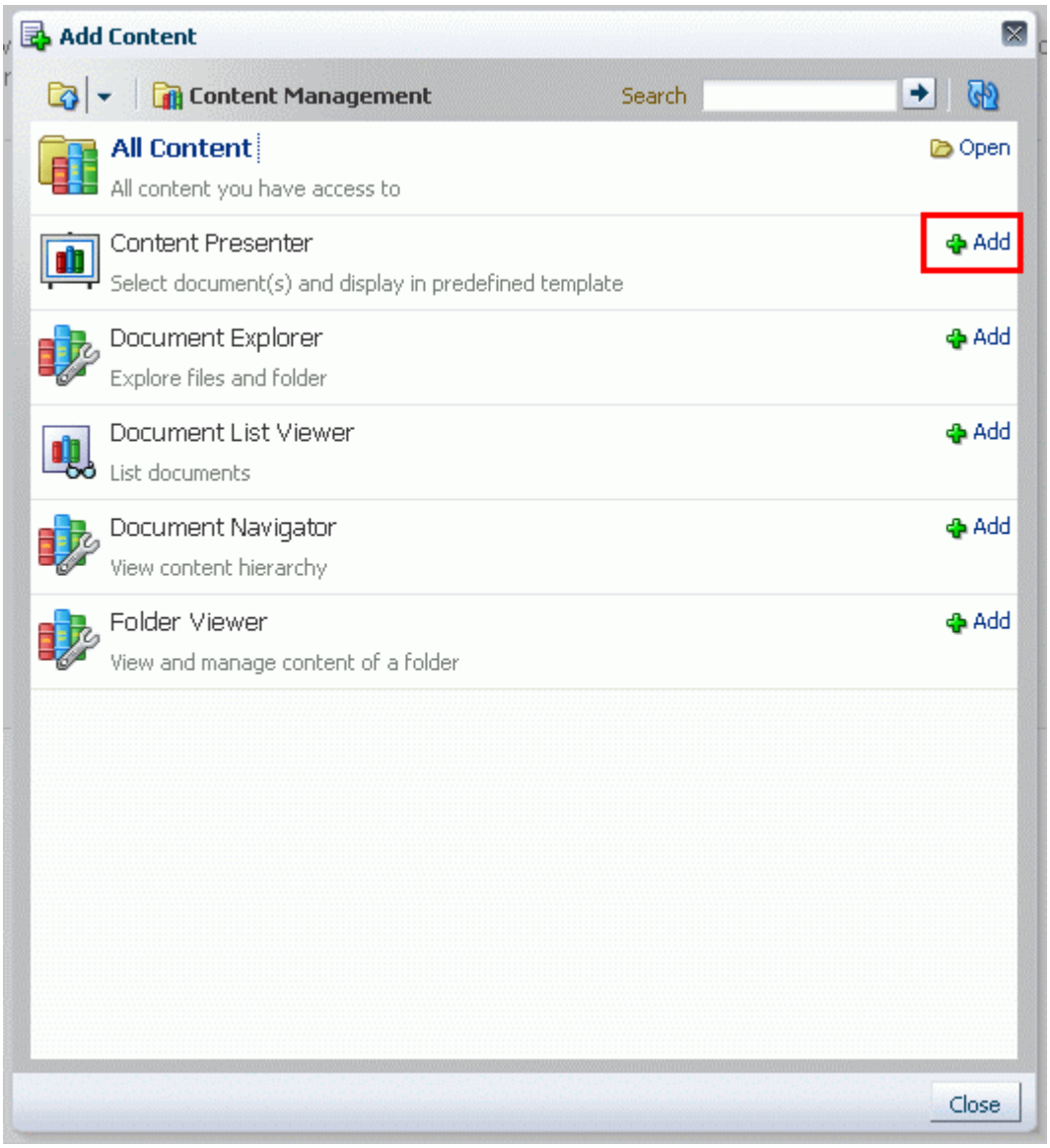
       If you are on the Administration Console, click **Back to Portal**, navigate to the Adobe page and select **Edit** from the pop-up menu.

b. Add the Content Presenter task flow.

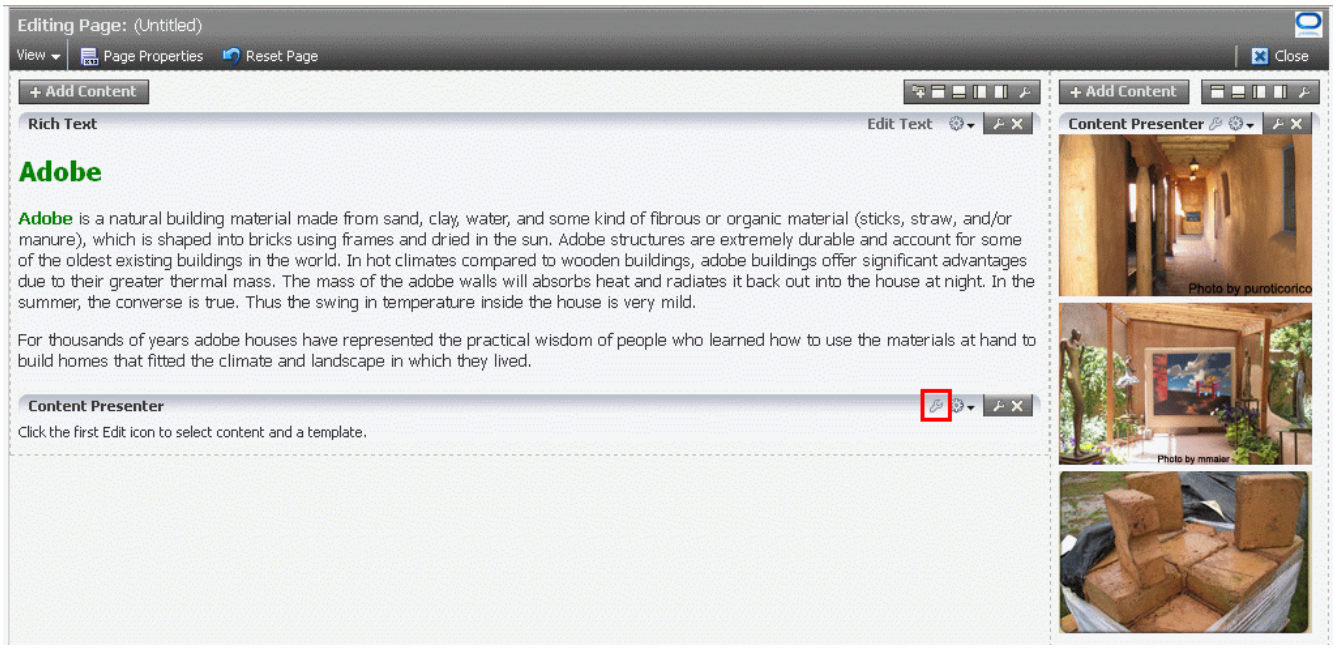
When the page is in Edit mode, click **Add Content** button in the left column.



In the Resource Catalog, open the Content Management folder and add the Content Presenter task flow to the page.



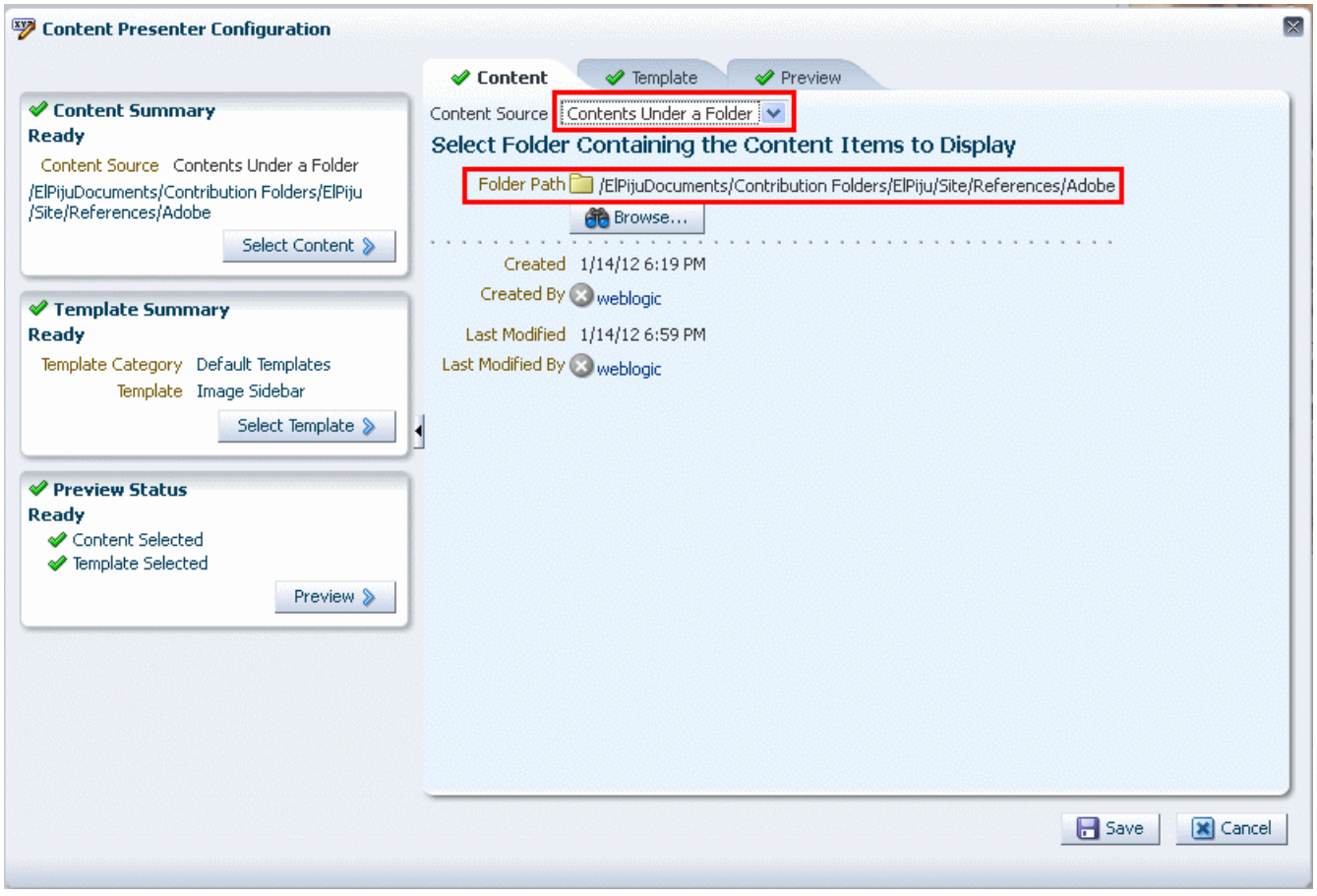
Close the Resource Catalog. The new task flow appears on the top of the left column. Drag it to the bottom of the column.



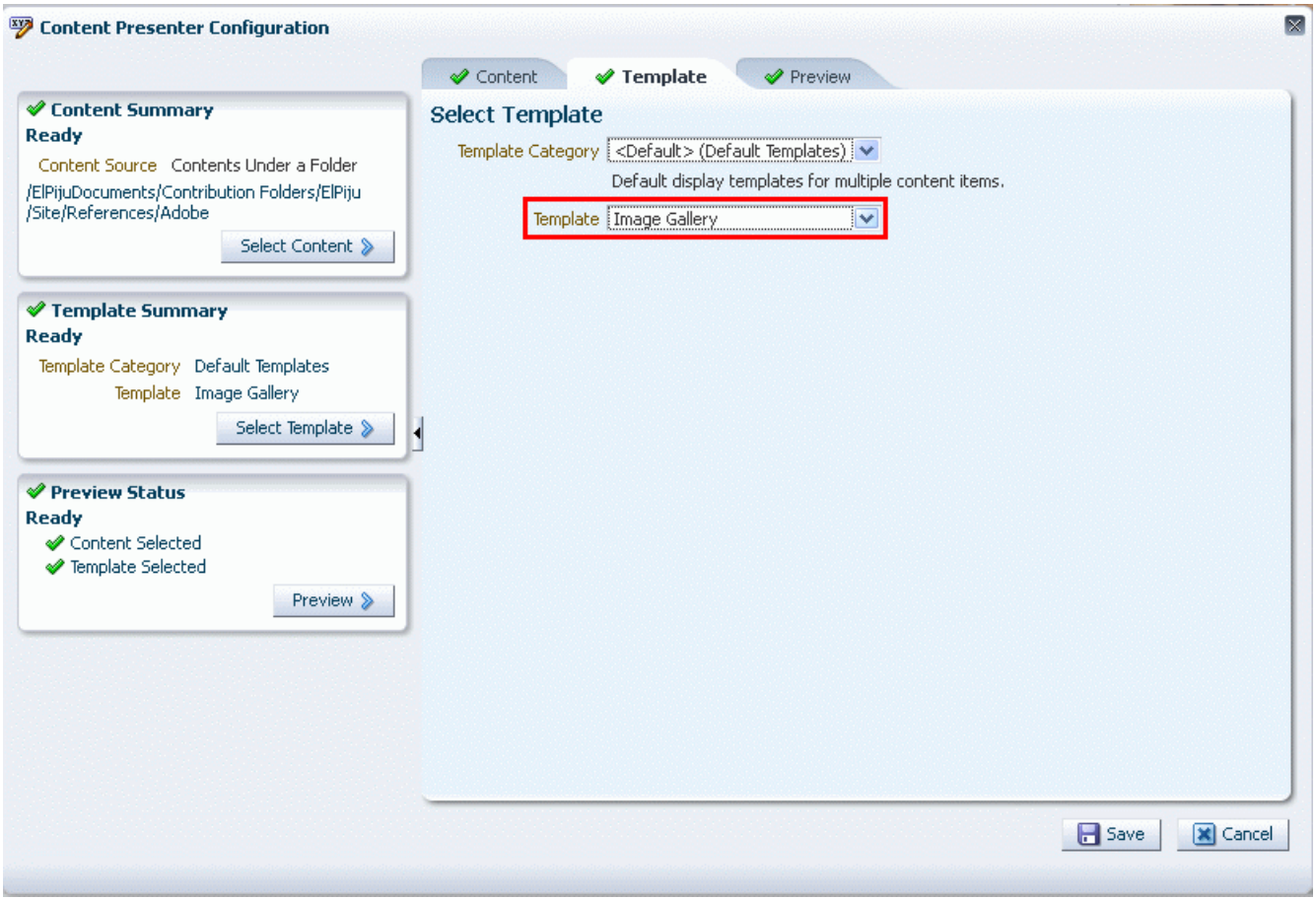
c. Configure the Content Presenter task flow

To configure the task flow, click the first Edit icon. In the Content tab choose *Contents Under a Folder* from the Content Source drop-down list. Click Browse to select a folder. Select the folder:

`/ElPijuDocuments/Contribution Folders/ElPiju/Site/References/Adobe`



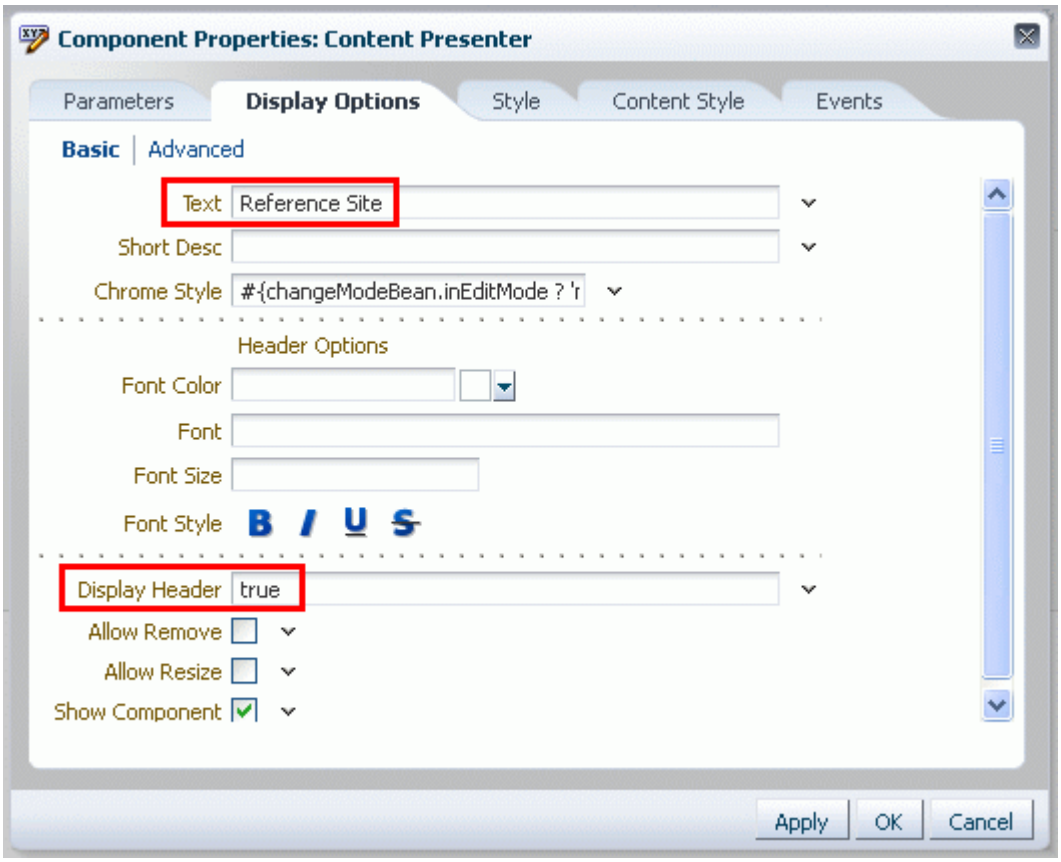
Open the Template tab and select **Image Gallery** for the template. Now you can click **Save** to save the settings and close the configuration window.



Although the page is still in Edit mode, you should already see the thumbnail images in the bottom of the left column.

d. Change the task flow title.

Optionally you can add a title to the thumbnail images. Click the Second Edit icon.



You can close the Edit mode. Click the Close link in the top right corner of the page.

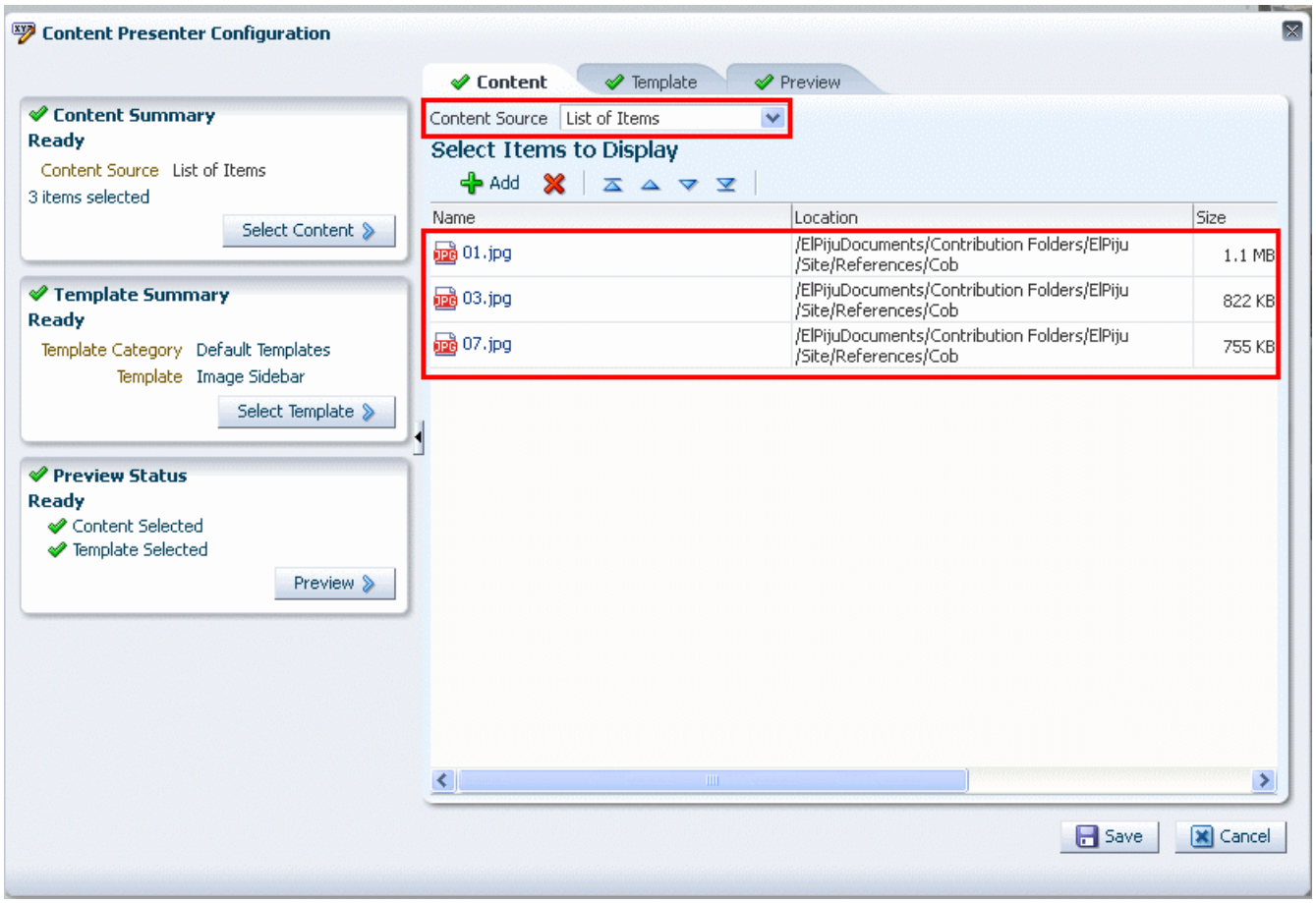
e. This is the final page. Test the image gallery clicking on a thumbnail and use the browse buttons to scroll through the images.

The next screenshot show the first image displayed. Note the close and next icons. Note also the title of the image: *Image 1 / 5 – Traditional adobe house – view from the south*. Remember how the title was created: the first part; *Image 1 / 5* – is created by fancyBox, the text comes from the comment attribute of the image. If comment is empty, it will display the file name instead.

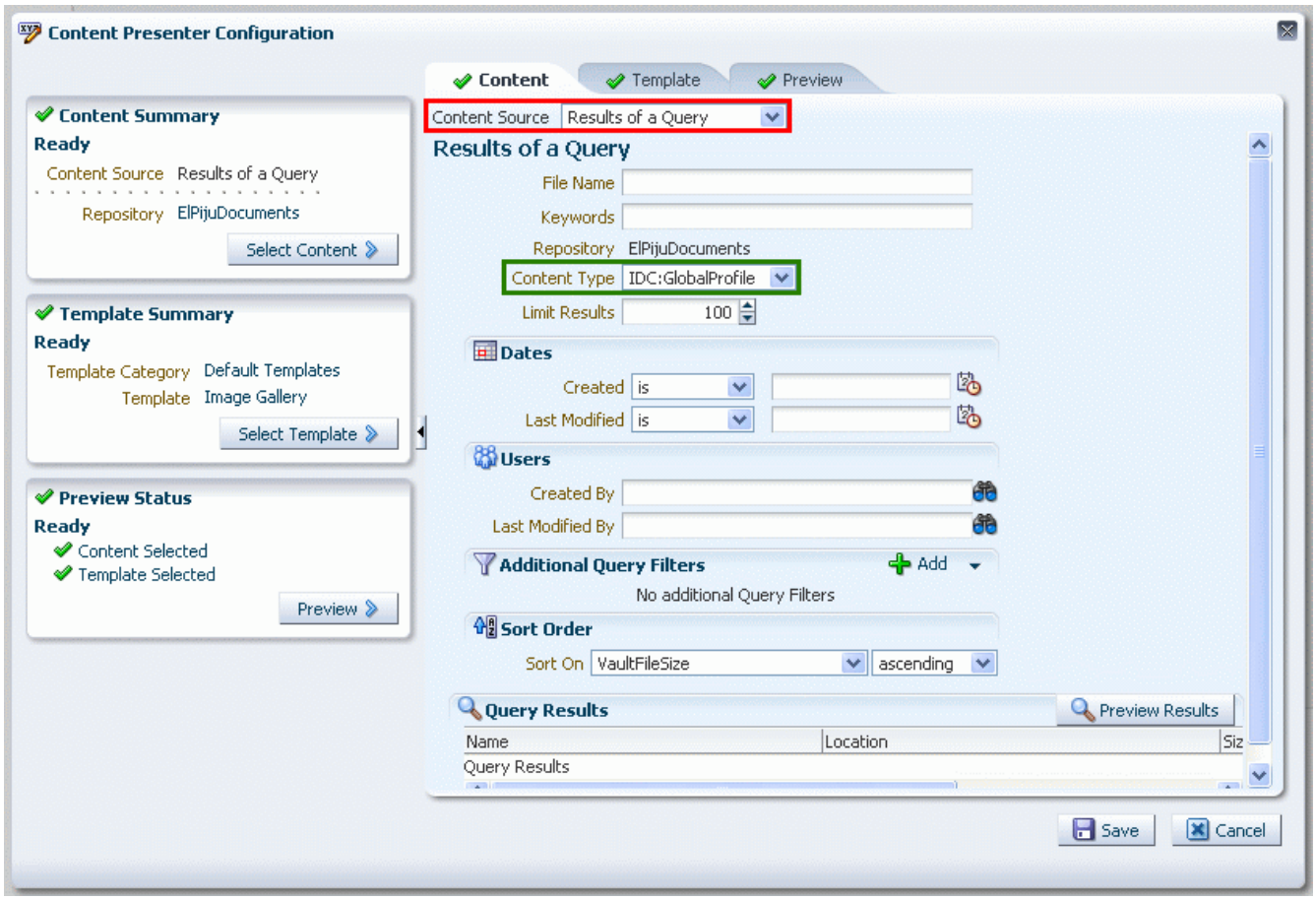
Notes on configuring the Content Presenter task flow:

In step c you have selected *Contents Under a Folder* for Content Source and defined the Folder Path. Alternatively:

- You can choose **List of Items** for Content Source and add a list of individual files. Those files do not have to come from the same folder.



- You can choose Result of a **Query for Content Source** and construct a query using the input fields. Note: if you select a specific content type, you can define Additional Query Items and Sort Order using the content type's attributes.



Merge Portal Resources

Recall that when you exported a portal resource, JDeveloper created an EAR file, which contains the `transport.mar` MDS archive. In this archive JDeveloper packed all of the necessary files:

- the JSF page fragment that implements the template which resides under `/oracle/webcenter/portalapp` folder,
- optionally the content of the shared folder and its subfolders,
- the `generic-site-resources.xml` file which resides under `\oracle\webcenter\siteresources\scopedMD\`*`unique_ID`*`\` folder.

The `generic-site-resources.xml` file describes a single resource, in our case a Content Presenter template. In the following example the resource description tag is displayed in bold:

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<resources
xmlns="http://xmlns.oracle.com/webcenter/portalframework/genericSiteReso
urces"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <resourceType name="siteTemplate" resourceBundle=""/>
  <resourceType name="pageStyle" resourceBundle=""/>
  <resourceType name="contentPresenter" resourceBundle="">
     <resource contentDir="/oracle/webcenter/portalapp/shared/"
               createdBy="system"
```

```
                createdDate="2012-01-24T14:06:07.312+01:00"
                description=""
                displayName="Image Gallery"
                guid="gsr83d29b25_e90d_4f8b_95b2_25e7d4d906f8"
   jspx="/oracle/webcenter/portalapp/contenttemplates/ImageGallery.jsff"
                modifiedDate="2012-01-24T14:06:07.578+01:00"
                seeded="false"
                type="contentPresenter"
                usesCustomSecurity="false"
                version="11.1.1.4.0"
                visible="TRUE">
          <customAttributes>
             <customAttribute name="template-type" seeded="false"
                              value="list"/>
             <customAttribute name="view-id" seeded="false"
                         value="elpiju.content.templates.imagegallery"/>
             <customAttribute name="category-id" seeded="false"
            value="oracle.webcenter.content.templates.default.category"/>
             <customAttribute name="category-name" seeded="false"
                              value="Default Templates"/>
             <customAttribute name="category-is-defaultview"
                              seeded="false" value="false"/>
             <customAttribute name="category-is-default" seeded="false"
                              value="true"/>
          </customAttributes>
       </resource>
   </resourceType>
   <resourceType name="resourceCatalog" resourceBundle=""/>
   <resourceType name="navigation" resourceBundle=""/>
   <resourceType name="taskFlow" resourceBundle=""/>
   <resourceType name="taskFlowStyle" resourceBundle=""/>
   <resourceType name="skin"
resourceBundle="oracle.webcenter.webcenterapp.resource.WebCenterResource
Bundle"/>
   <resourceType name="dataControl" resourceBundle=""/>
</resources>
```

Although currently it is not supported by any tools, you could easily merge more resource archives.

- Unzip the `transport.mar` files to separate root folders. You can use any program that handles ZIP archives, including Java's jar utility.
- Copy the content of all the `/oracle/webcenter/portalapp` folders together to the first root folder.
- Edit the `generic-site-resources.xml` file under the first root folder. Add the resource tags from the other `generic-site-resources.xml` files to this file. Note that you might merge different types of resources to the same archive, just make sure, you add the `<resource>` definition inside the appropriate tag, one of the following:

```
 <resourceType name="siteTemplate" ...> ... </resource>
 <resourceType name="pageStyle" ...> ... </resource>
 <resourceType name="contentPresenter" ...> ... </resource>
 <resourceType name="resourceCatalog" ...> ... </resource>
```

```
<resourceType name="navigation" ...> ... </resource>
<resourceType name="taskFlow" ...> ... </resource>
<resourceType name="taskFlowStyle" ...> ... </resource>
<resourceType name="skin" ...> ... </resource>
<resourceType name="dataControl" ...> ... </resource>
```

- Zip the first root folder to a file called `transport.mar`.
- Zip this file to an EAR file.

Now you have a combined resource archive. The next time you import this archive into a JDeveloper project at design time, or into a running WebCenter Portal application, it imports all of the resources at once.