ORACLE
FUSION MIDDLEWARE
WEBCENTER

Migrating an Oracle WebCenter Portal Framework Application to the Oracle WebCenter Portal Server

Part 1 – Migrating an application and its page and reusable components

ORACLE

Table of Contents

Introduction

Oracle WebCenter Portal Server is a modern web platform that enables building enterprise self-service portals and composite applications. Portals and applications are built in the browser using both seeded and custom components. Components are developed in JDeveloper or created with browser-based tools. The WebCenter Portal Server enables a clear separation between the skills required for developing custom portal assets and those required for building portals or composite applications at runtime. This separation is achieved by leveraging assets and components that are available in the portal resource catalogue. Separation enables the pairing of specific expertise with relevant sets of portal tools, empowering business users, developers, and portal administrators to apply their skills and talents to rapid development of portal solutions. The result: portals that deliver self-services to customers, employees, or partners in a highly secure and personalized way through multiple channels.

Oracle WebCenter Portal Server is the evolution of WebCenter Spaces technology. In WebCenter Portal 11.1.1.4.0 release we added Portal Framework technology for developing Portal Framework application in JDeveloper. The WebCenter Portal Framework technology is still available in the product and remains fully supported in the 11g R1 product line. However, it is in maintenance mode since the WebCenter Portal 11.1.1.7.0 release. This means that Oracle has stopped adding new features to it. For example, Portal Builder, mobile web support, and other new features added in WebCenter Portal Server 11.1.1.8.0 release are not available in the Portal Framework.

The WebCenter Portal Framework will not be available in the WebCenter Portal 12c R2 release. Oracle recommends that customers who are running Portal Framework application and want to benefit from using new features and capabilities consider migration from Portal Framework to Portal Server. We also discourage customers who are planning to start any new development on WebCenter Portal 11g R1 (11.1.1.8.0 onward) from using Portal Framework. We recommend such customers use Portal Server technology instead.

This white paper outlines an approach and provides recommendations for migrating Portal Framework applications to the WebCenter Portal Server. Given the variety and complexity of custom-built solutions, providing an automated migration path is not feasible. However, since the underlying technology is the same, migration can be successfully completed following the recommendations described in this white paper. Generally, the more modular a Portal Framework application is, the easier it is to migrate to WebCenter Portal Server. When planning and executing the migration, we also suggest that you address the following functional areas separately—

» Pages and Navigation
» Custom code, such as task flows, backing beans, data controls, and related artifacts
» Assets, such as page templates, skins, page styles, and content presenter templates
» Services data, such as data from discussion forums and content on the content server
» The application security model

This white paper covers recommendations for migrating pages, page navigation, and custom code. Subsequent white papers will cover the other relevant areas.

Assumptions

This white paper assumes that you are familiar with the portal creation experience and page security model in the Portal Server.

Migration Overview

A Framework application employs modular artifacts, such as task flows, or artifacts that are intertwined into the application, such as a JEE filter. For the most part, Framework application artifacts can be mapped to their portal equivalents. However, during migration, bear in mind that, apart from mapping the artifacts to portal equivalents, some rethinking of the overall approach may be necessary. A detailed explanation with examples follows.

TABLE 1. FRAMEWORK APPLICATION TO WEBCENTER PORTAL ARTIFACT MAPPING TABLE

Framework Application	WebCenter Portal	Comments
The application	A portal	A portal provides containment to related functionalities and application flows. If the application has multiple functionalities or user groups that distinctly consume that functionality, consider creating multiple portals, each serving a specific purpose. This helps in building a modular portal that is easier to maintain. Example: If you created multiple applications for various functions such as HR and Finance, consider creating a portal for each of these functions. It could also be that you have a single application with a mechanism for navigating to these functions. If you have self-service applications, consider creating a cohesive experience for the end-user by aggregating them on a portal with reusable task flows.
Task Flows, Data controls, connections, backing bean, business logic	Deployed as shared libraries by extending WebCenter Portal	ADF task flows are reusable components that can be consumed on an ADF page. ADF task flows can be consumed as is in WebCenter Portal. ADF connections—such as web services and URL connection—can be recreated on WebCenter Portal Server using Enterprise Manager or the relevant WLST commands.
Pages & navigations	Pages	Pages map one to one on the portal server. If a set of pages provides a related functionality, consider creating a set of sub-pages below a main page.

		If application functionality is coded on a page, consider creating a task flow out of it. This allows highly modular design, and the task flows can be reused on pages across portals. Security, in the form of page access, must be migrated to the Portal Server with care. Migration will likely break some navigation links on pages, and these must be fixed.
Assets	Assets at portal or global level	Typically, assets—such as page templates, skins and content presenter templates—can be migrated using JDeveloper. However, care should be taken in deciding whether the assets can be owned by the portal or promoted to Shared Assets. The latter helps share the assets across portals. For example, corporate page templates should be a shared asset so that other portals can use them as well.
Services data	Data Sources to the services' schema	Migrating the services data includes creating a data source that points to the services' schemas in the WebLogic Server console. Depending on a service, it can involve creating an appropriate ADF connection to the underlying service.
Security model	Security configuration at portal or global level	Custom roles & permissions When you have an application that is accessible to users who belong to a certain role, consider replicating that role at the portal or global level.
Developing for WebCenter Portal & Project Lifecycle Management	Oracle JDeveloper & WLST scripts and build tools	Oracle JDeveloper has extensions that help build WebCenter Portal assets and deploy them to the portal server. You can use WLST commands to manage the lifecycle of artifacts, such as assets, portals, and backup.

It is evident that many artifacts can be migrated with relative ease, while some others must be reimagined. Consider carefully the choices you make when you migrate a Portal Framework application to the WebCenter Portal platform.

Migrating the Framework Application - First Step

The Framework Application should be imagined as a portal on the WebCenter Portal server. In this section, we will see how an application can be mapped to a portal. Further, the pages and navigations in the Framework Application can be re-created on the newly created portal with the following steps:

1. Create a portal.
2. Re-create page navigation model.
3. Migrate any dynamic navigation elements into the portal.

These steps will ensure that you create a portal that duplicates the Framework Application along with its exact page structure.

Creating a Portal

The first task is to define your actual navigation model in a format that you can use to create a new portal. In this way, you will have a seeded navigation and set of pages that correspond to your Portal Framework.

Using a text editor, replicate your navigation model. Create entries for each page using the syntax defined in the Creating Pages When Creating a New Portal. For example:

```
Bank
+Checking
```

```
++My Checking
++My Debit Cards
+Savings
++Regular Savings
+CDs
++Standard Term
+Credit Cards
+Student Banking
+Online Banking
Borrow
+Credit Cards
+Mortgage
Invest
+Retirement and IRAs
++Roth IRA
+Trading
+Fixed Income & Bonds
Research
+Quotes
+Alerts
+Stocks
+Learning Center
++Investment Products
+++Mutual Funds
+++Investing
+++Personal Savings
+++Mortgages
```

Log on to WebCenter Portal as a user with "Create Portal" permission, and navigate to—

`http://<server>:<port>/webcenter/portal/builder/portals`

Select **Create Portal**—

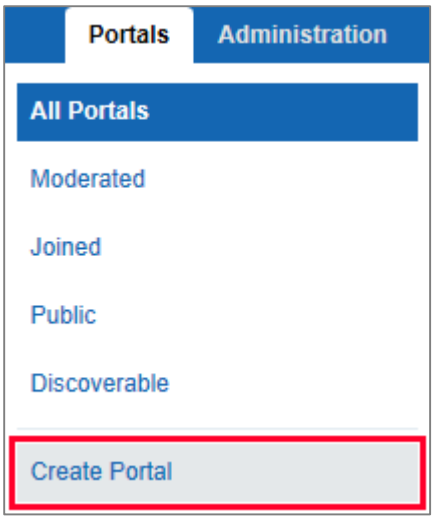

Figure 1. Create a portal

Click the **Use This** button to select the standard Portal Template.

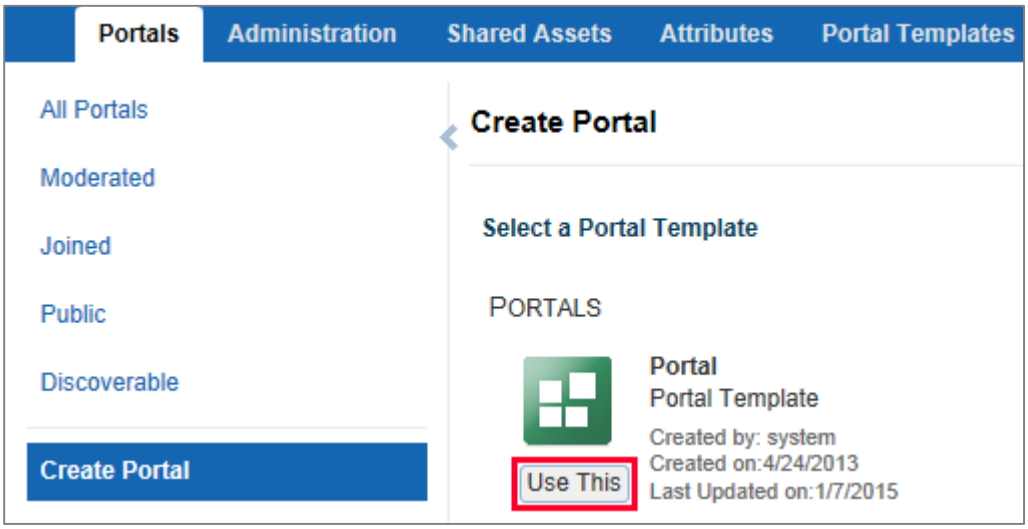

Figure 2. Select the Portal template to create a portal

Enter the name of your Framework Application, and click **Add Pages**. Note that the URL field is updated to the same value.

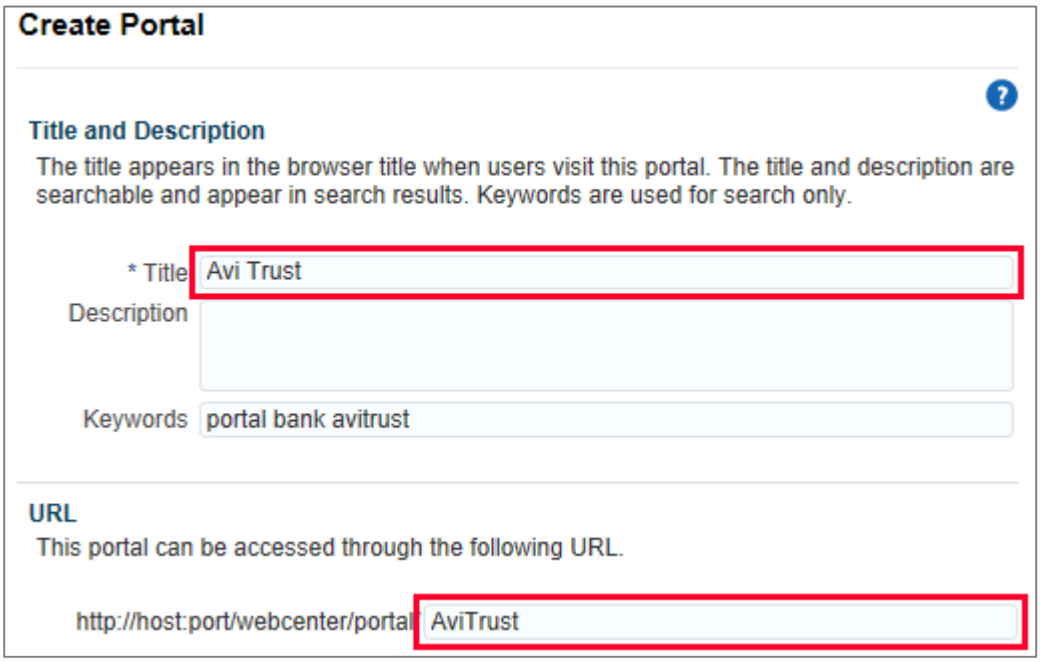

Figure 3. Name the portal as 'Avi Trust' and specify URL

Copy your formatted list of pages and paste it into the Add Pages field (Figure 4). Click **Create**.

Note: The pages are created with the default page style while your framework application page may use a specific page style. However, it should not matter as in the next step, you will copy over the page code into the portal page and bring along the page style as well as its content. The purpose of this step is to conveniently create the page hierarchy along with the portal.

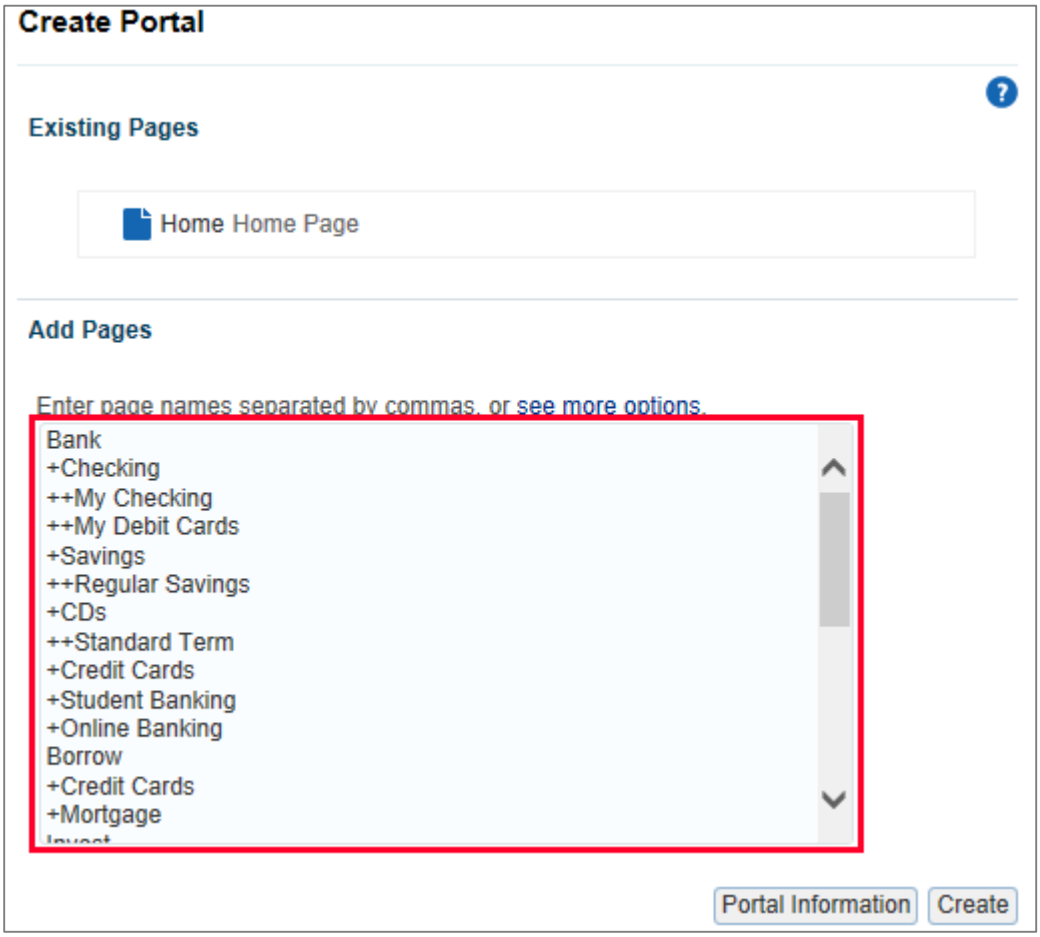

Figure 4. Specify the portal's pages and their structure in "Add Pages"

After portal creation completes, click **View your portal** to review your portal

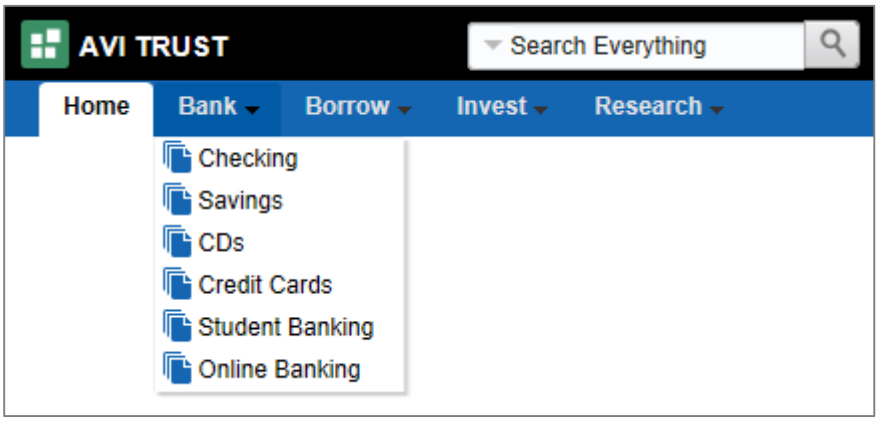

Figure 5. The portal's page structure after it is created

The default page template for a new portal supports only two levels of navigation. To see all the levels of navigation and pages created for your portal, press "<ctrl><shift>e" to enter edit mode. The entire portal structure appears on the left, and you can select a page to edit it directly.

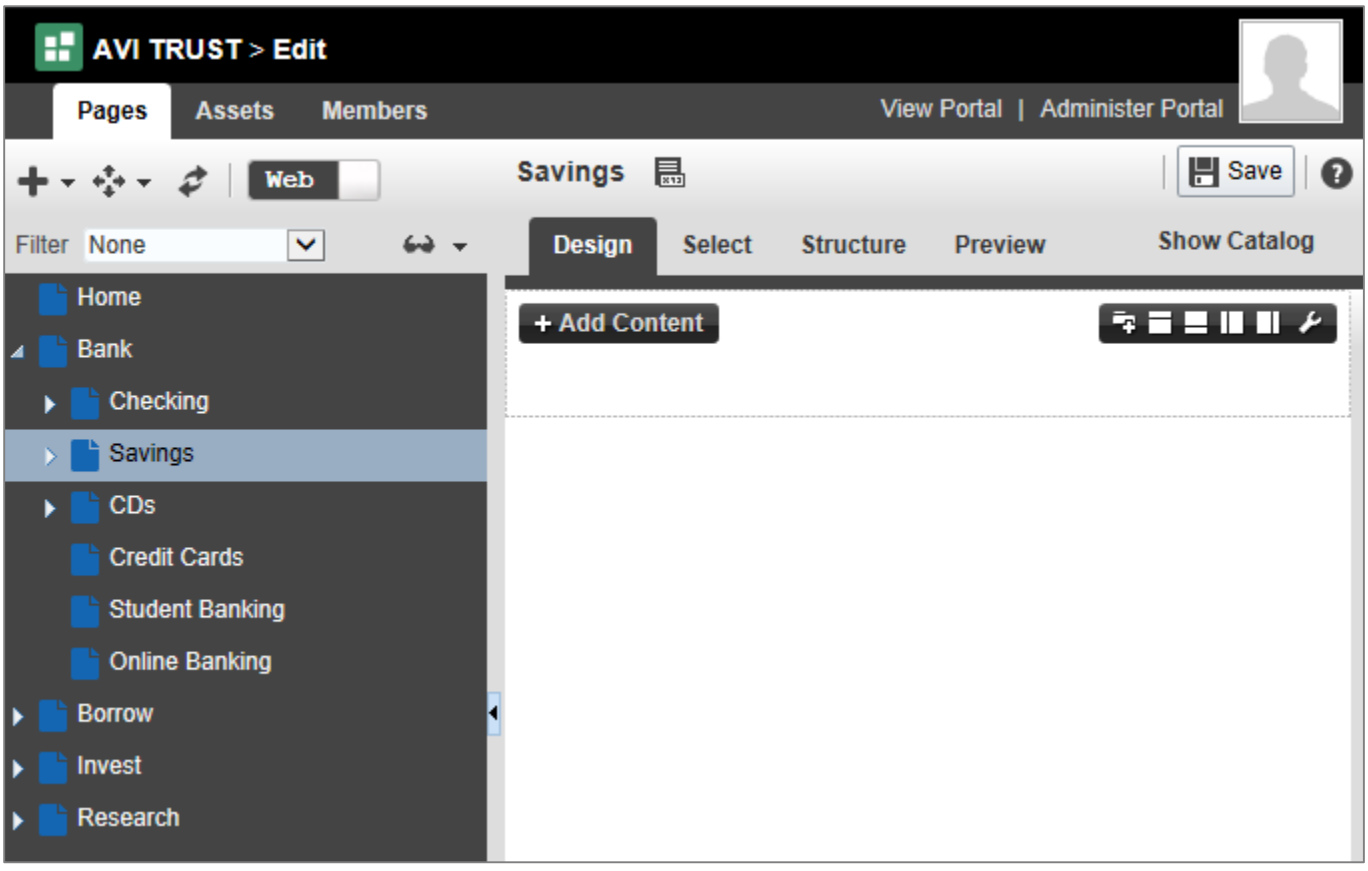

Figure 6. Portal in Edit mode with the pages specified during its creation

Summary

The core structure of your Portal Framework application is now available on the Portal Server. You're ready to migrate your portal assets—custom code, ADF skin, page templates, page styles, and content presenter templates—to the Portal Server.

More Information

More information about creating a portal is available in Creating and Building a New Portal.

Migrating Pages

In the previous section, you built the page structure on a portal that represents your Portal Framework application. The next step is to migrate page content from your application into the portal.

The pages you intend to migrate will likely have various levels of complexity. Some pages may contain component "instance data" that must be copied when the page is copied. Others contain references to components, such as documents, that have been separately migrated and may need to be fixed up.

The steps for migrating pages are:

1. Migrate the page and page definition files.
2. Migrate parameters for the page
3. Migrate references to data, such as document IDs in the WebCenter Content repository that may have been migrated.
4. Migrate references to instance data, such as portlet preferences or task flow instance level customizations that are captured outside of the page.

Note: Security is managed as a separate step and isn't handled as part of page migration.

Depending on how complex the page hierarchy of your framework application is, you can either copy the pages, and page definitions of individual pages using the portal administration screen or use WLST commands and some scripting.

- If your application has few pages, consider using the Portal Administration screen. This is explained in the Migrate the page and its page definition using portal administration section.
- If your application has many pages, populating the pages on the Portal Server may become tedious. In this case, consider using WLST commands. This is explained in next section. Consider rethinking your page hierarchy and see how it can be categorized or modularized so that pages can go into different portals instead of into a monolith portal.

Migrating Pages and Page Definitions Using WLST Commands

Using MDS WLST commands is a faster way to re-create Portal Framework Application pages in a portal. However, this too involves some manual steps to ensure that the Portal Framework Application page content is correctly exported to the corresponding page in the portal. To migrate pages using WLST commands, follow these steps:

1. Export pages from the Portal Framework Application's MDS repository.
2. Export portal pages you created on the WebCenter Portal server from its MDS repository.
3. Copy the framework application's page/page definition source to the corresponding page in the portal.
4. Import portal pages back into MDS repository.

By default, framework application pages created at runtime reside in the `/oracle/webcenter/portalapp/pagehierarchy` MDS namespace. The location of the pages created at

design time in JDeveloper is based on your application. Typically, the location is `/oracle/webcenter/portalapp/pages`. The first step to re-create these pages in a portal is to export pages from both the framework application and the portal using MDS WLST commands—

Export Portal Framework Application Pages

```
$ cd ${FMW_HOME}/common/bin/
$ ./wlst.sh
wls: > connect('user', 'password', 't3://host:port')
wls: > exportMetadata('frmk_app_name','managed_server', '/frmk_pages',
docs='/oracle/webcenter/portalapp/pagehierarchy/**')
wls: > exportMetadata('frmk_app_name','managed_server', '/frmk_pageDefs',
docs='/pageDefs/oracle/webcenter/portalapp/pagehierarchy/**')
```

Where:

- `user`, `password` refer to the credentials of the WebLogic Server admin user
- `host`, `port` refer to the host name and port number of the WebLogic Server Admin server that manages the server where the framework application is deployed.
- `frmk_app_name` refers to the name of your Portal Framework Application.
- `managed_server` refers to the name of the managed server where your framework application is deployed.
- `/frmk_pages` refers to the location on the file system where you want the MDS WLST command to export the files into.

Note that you may have created pages in your application in locations other than the recommended namespace `/oracle/webcenter/portalapp/pages`. In that case, you should include those pages' namespaces in the migration process.

Export Portal Pages

Repeat the same set of commands to export the pages and page definitions of the portal you created on the WebCenter Portal server:

```
$ cd ${FMW_HOME}/common/bin/
$ ./wlst.sh
wls: > connect('user', 'password', 't3://host:port')
wls: > exportMetadata('webcenter','WC_Spaces', '/portal_pages',
docs='/oracle/webcenter/page/scopedMD/<Portal_GUID>/**')
wls: > exportMetadata('webcenter','WC_Spaces', '/portal_pageDefs',
docs='/pageDefs/oracle/webcenter/page/scopedMD/<Portal_GUID>/**')
```

Where:

- `user`, `password` refers to the credentials of the WebLogic Server admin user.
- `host`, `port` refers to the host name and port number of the WebLogic Server Admin server that manages the WebCenter Portal server.
- `webcenter` refers to the application name of WebCenter Portal.
- `WC_Spaces` is typically the name of WebCenter Portal managed server.
- `/portal_pages` refers to the location on the file system where you want the MDS WLST command to export the files into.
- `/Portal_GUID` refers to the Internal ID (GUID) of the portal you created on the portal server.

You can get the internal ID of a portal from its **Overview** tab when administrating a portal.

Copy Portal Framework Application Pages to the Portal

Map the page/page definition of the framework application pages to the corresponding pages created in the portal. Ensure that you copy the contents of the files correctly from one to the other (ex. From `frmk_pages` to `portal_pages`).

After copying the contents of the framework application pages to the portal pages, you can import the pages back to the portal server. However, before executing the steps, you should fix up Navigations and other artifacts on the pages because the IDs would have changed. This is detailed in the Fix Navigations on pages section.

The `exportMetadata` command also exports the page/page definition customizations. You should ensure that you fix the MDS customization layer name and its value so that the customizations continue to work in the Portal Server. This is explained in the Migrating Task Flow and Page Customisations section.

Commands to import pages and page definitions back to portal server:

```
$ cd ${FMW_HOME}/common/bin/
$ ./wlst.sh
wls: > connect('user', 'password', 't3://host:port')
wls: > importMetadata('webcenter','WC_Spaces', '/portal_pages',
docs='/oracle/webcenter/page/scopedMD/<Portal_GUID>/**')
wls: > importMetadata('webcenter','WC_Spaces', '/portal_pageDefs',
docs='/pageDefs/oracle/webcenter/page/scopedMD/<Portal_GUID>/**')
```

Migrate a Page and Its Page Definition Using WebCenter Portal Administration

Migrating each page involves the following steps:

1. Migrate page .jspx and page definition files.
2. Add parameters to the page.
3. Update pages to use migrated data.
4. Update pages to use migrated instance data.

You can migrate simple pages relatively easily since only the .jspx and page definition files are moved. Since there is a one-to-one mapping between these and the pages you pre-created when the portal was created, the effort is only in merging the content of these files.

Navigate to the pages in your new portal on the Portal Server:

`http://<host>:<port>/webcenter/portal/builder/portals/admin/MyCustomAppPortal/pages`

Expand the list of pages on the left, which follow your navigation model structure, and select the page you want to migrate

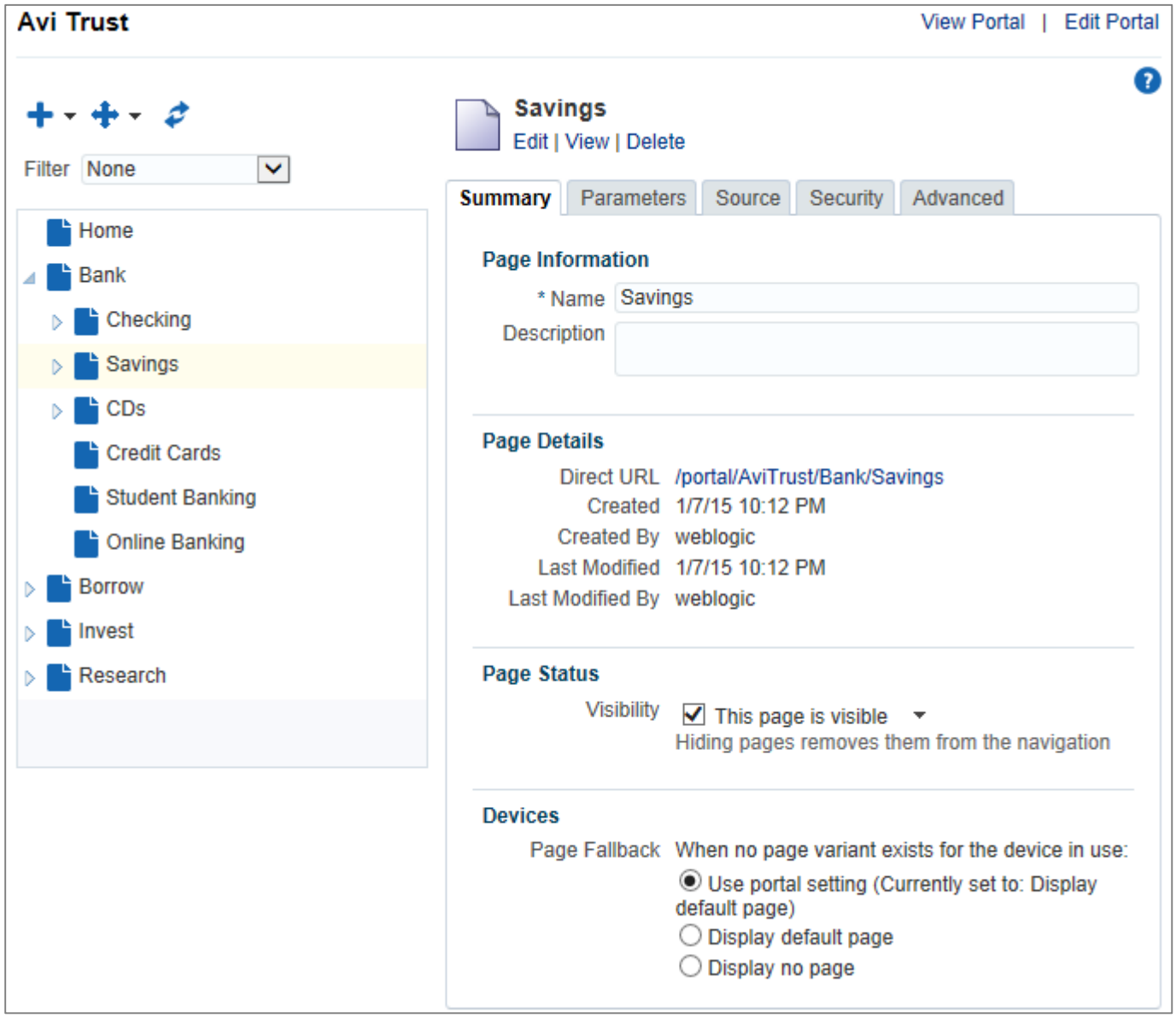

Figure 7. Summary of **Savings** page in the portal

Click the **Source** tab to see the Page and PageDef sub-tabs at the bottom.

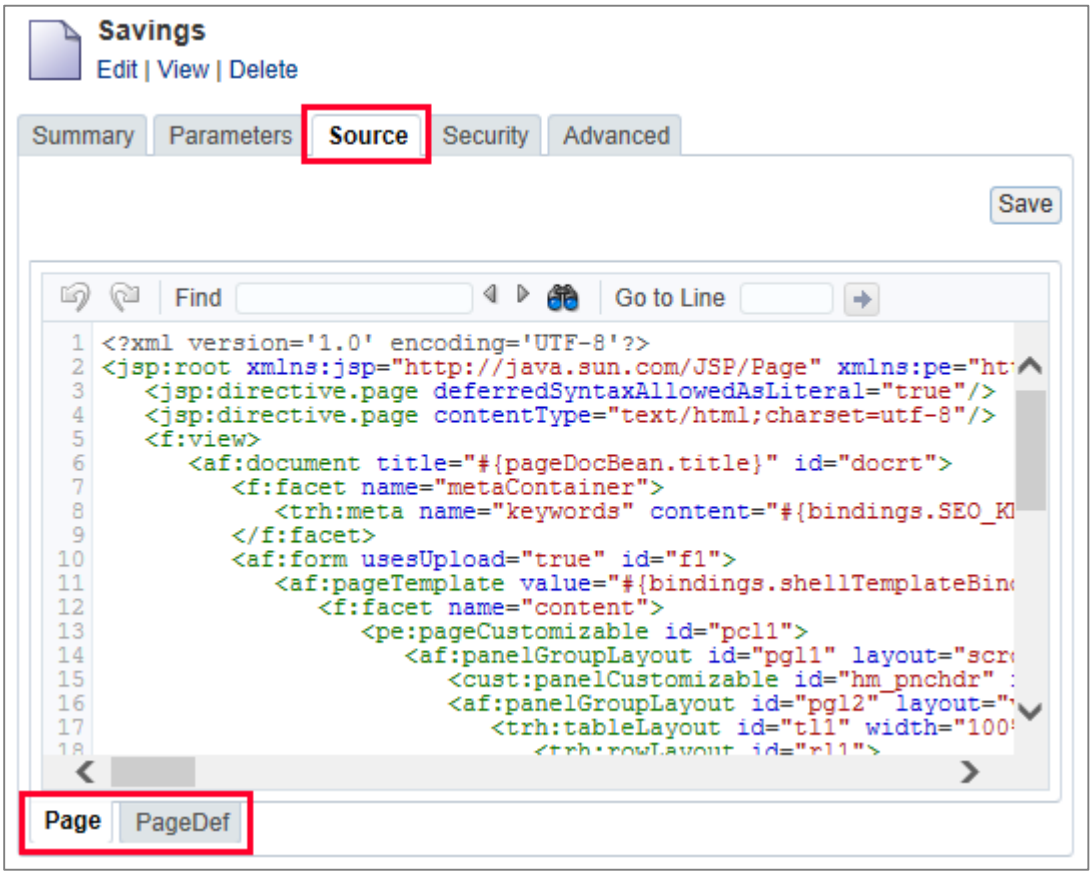

Figure 8. Source tab of **Savings** page in the portal

» The **Page** tab refers to the .jspx page from your Portal Framework application.

» The **PageDef** tab refers to your PageDef.xml file for your page in your Portal Framework application.

The merge for the .jspx page will vary, but typically you should only need to do the following—

Add any additional namespace entries:

```
<jsp:root xmlns:jsp="http://java.sun.com/JSP/Page" version="2.1"
          xmlns:f="http://java.sun.com/jsf/core"
          xmlns:h="http://java.sun.com/jsf/html"
          xmlns:af="http://xmlns.oracle.com/adf/faces/rich"
          xmlns:pe="http://xmlns.oracle.com/adf/pageeditor"
          xmlns:cust="http://xmlns.oracle.com/adf/faces/customizable">
```

Replace the "`content`" facet with the Portal Framework Application's page's contents, leaving out the template code:

```
<f:facet name="content">
  <!-- Your page content goes here -->
</f:facet>
```

Click the **PageDef** tab to view the default Page Definition

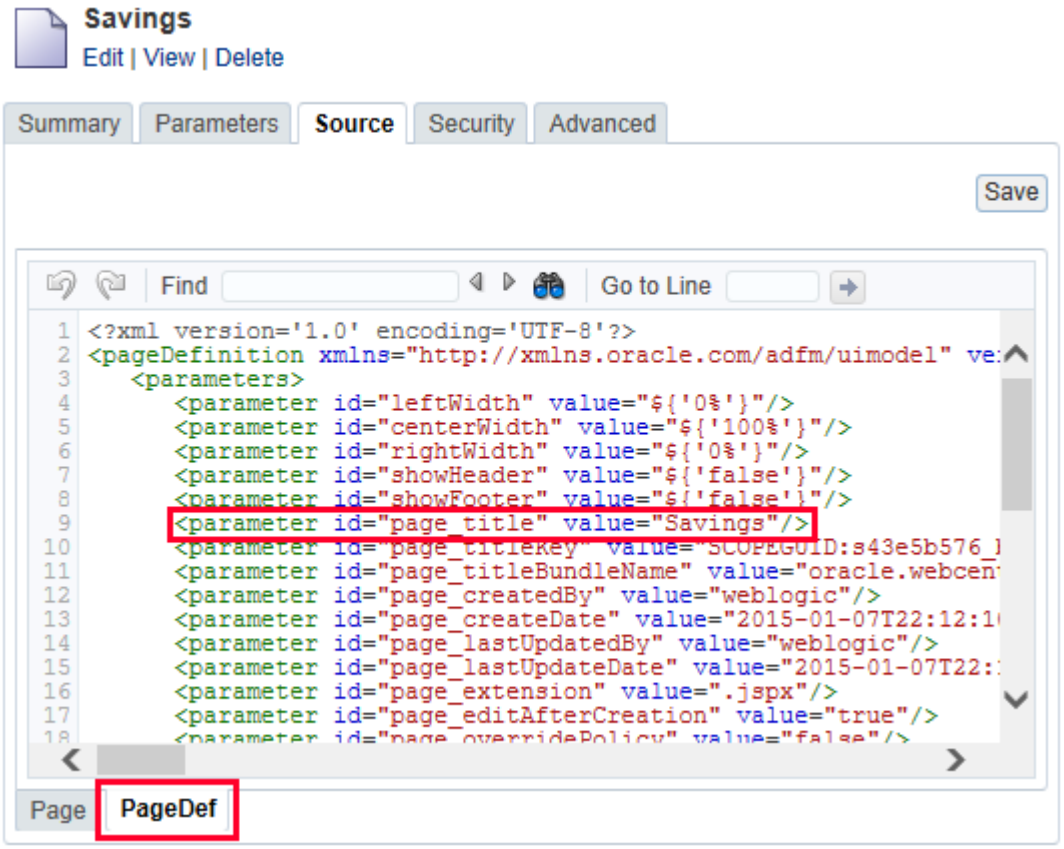

Figure 9. PageDef tab showing 'Savings' page's page definition

Note: The portal server stores information about the parameters and permissions in the PageDefinition under `<parameters>` and `<permission>` nodes respectively. Do not change these seeded values. You will only change the `<executables>`, `<bindings>` and, `<eventMap>` sections.

Update the `<executables>` section of the page definition in the Portal Server.

Add all the elements from the `<executables>` section of your Portal Framework application page's page definition, making sure to keep the `<page>` element from the portal server page definition. The `<page>` element specifies the page template for the page.

Add all the elements into the `pageDef`:

```
<executables>
  <page id="shellTemplateBinding" Refresh="ifNeeded"
viewId="#{WCAppContext.application.siteTemplatesManager.currentSiteTemplateViewId}"
/>
  <variableIterator id="variables">
    <variable Type="java.lang.String" Name="handleLogin_username"
```

```
              IsQueriable="false"/>
    <variable Type="java.lang.String" Name="handleLogin_password"
              IsQueriable="false"/>
  </variableIterator>
  <taskFlow id="pageeditorpanel"
            taskFlowId="#{pageEditorBean.pageEditorPanel}"
            activation="deferred"
            xmlns="http://xmlns.oracle.com/adf/controller/binding"/>
  <taskFlow id="test1" taskFlowId="/WEB-INF/test.xml#test"
            activation="deferred"
            xmlns="http://xmlns.oracle.com/adf/controller/binding"/>
</executables>
```

Update the `<bindings>` section of the page definition in the Portal Server.

Replace with the `<bindings>` content from your Portal Framework application:

```
<bindings>
  <methodAction id="handleLogin" RequiresUpdateModel="true"
                Action="invokeMethod" MethodName="handleLogin"
                IsViewObjectMethod="false" DataControl="LoginDC"
                InstanceName="LoginDC.dataProvider">
    <NamedData NDName="username" NDType="java.lang.String"
               NDValue="${bindings.handleLogin_username}"/>
    <NamedData NDName="password" NDType="java.lang.String"
               NDValue="${bindings.handleLogin_password}"/>
  </methodAction>
  <attributeValues IterBinding="variables" id="username">
    <AttrNames>
      <Item Value="handleLogin_username"/>
    </AttrNames>
  </attributeValues>
  <attributeValues IterBinding="variables" id="password">
    <AttrNames>
      <Item Value="handleLogin_password"/>
    </AttrNames>
  </attributeValues>
</bindings>
```

Add in any `<eventMap>` to the Page Definition.

Click “Save” on the Source tab

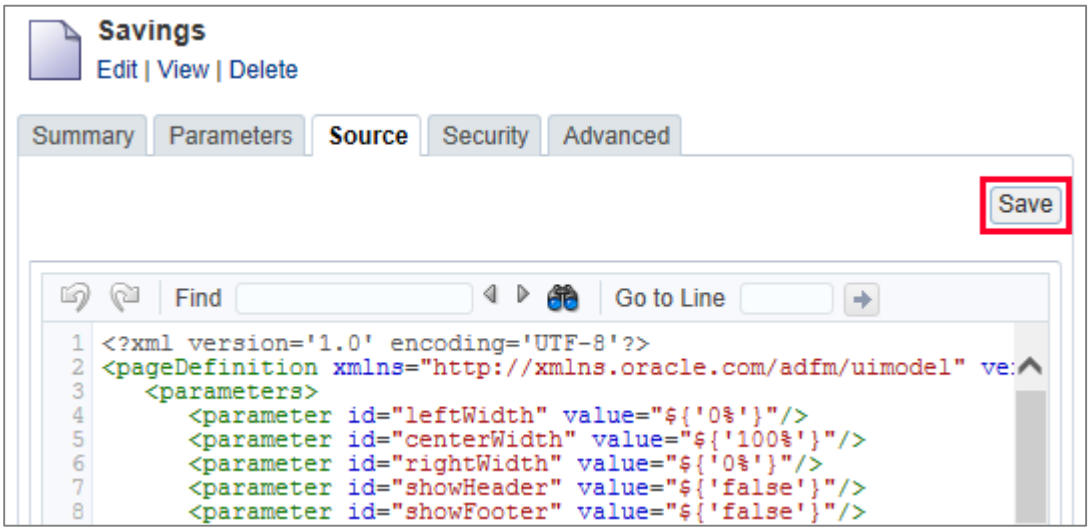

Figure 10. Save the page after editing its source

Summary

The steps described in this section assist with migrating pages from a Portal Framework Application to a Portal Server. When you use WebCenter Portal Administration, you can migrate only one page to the portal server at a time. You must repeat these steps for all pages in the custom application.

More Information

For more information on creating, editing, and managing portal pages see:

- Creating and Editing a Portal Page documentation
- Managing a Portal Page documentation
- Metadata Services (MDS) Custom WLST Commands

Migrating Custom-Built Components

Custom components refers to task flows, declarative components, and their associated artifacts such as backing beans, business logic, ADF connections, and data controls. In this section we will see how these components can be packaged and deployed into WebCenter Portal server. Here is a list of custom-built components:

1. Task Flows
2. ADF Connections & Data Controls
3. Backing Beans and Business Logic
4. Declarative Components and Custom Tag Libraries
5. JEE Filters & Servlets
6. Portlets

Developing for WebCenter Portal

JDeveloper IDE should be used to develop assets for WebCenter Portal. It provides powerful features for developing, testing, and deploying components based on Java, ADF, and other web technologies relevant to

WebCenter Portal. Consider using the methods described in Developing Components for WebCenter Portal Using JDeveloper to build the reusable components and assets described in the following sections.

Task Flows

ADF provides a mechanism to bundle a task flow as an "ADF Library", encapsulating the implementation including the UI views, backing beans, and business logic. To migrate the Portal Framework Application to portal server, consider reviewing your application for opportunities to move related functionalities into a task flow. This will modularize the implementation and aid in easier migration to portal server. If you already use task flows to build your application, it will be easier to migrate to a portal platform.

For example, imagine that your application gives employees the ability to raise service tickets using multiple pages. You should consider encapsulating this into possibly multiple parameterised task flows—one to raise service requests and one to view and manage them. Disparate pages and the backing code that implement this functionality will then be housed in one entity, a task flow. Ensure that a task flow uses data controls and connections if it relies on data from external sources to function use data controls in lieu of resorting to hard-coding the connection details. This will allow easier migration to the Portal Server, especially during stage-to-production lifecycle operations.

You might have built task flows in the application itself. In such cases, consider refactoring the task flows and moving them to separate applications as mentioned below. The namespace of the task flows in a Portal Framework Application is `oracle/webcenter/portalap/pagefragments`. Consider refactoring the namespace to something like `com/example/avitrust/tickets/…` This will provide for cleaner separation of your task flow namespaces vis-à-vis the namespaces used by the WebCenter Portal Server.

Create an application (`.jws`) in JDeveloper to group all the related task flows, with each of them in a separate project (`.jpr`). This will let you take advantage of JDeveloper's support to create, test, build/package ADF projects, and to use source control systems to manage your code base. Take care to ensure that the path of task flow definitions, views, and their page definition files are the same as they were before migrating them to a new JDeveloper application. This will ensure that the references to these task flows on pages will not break.

To summarise, reimagine your framework application as a collection of task flows that provide the related functionalities. We will discuss the ways to surface these task flows in WebCenter Portal in the "Extending Oracle WebCenter Portal Server" section.

More Information

» Getting Started with ADF Task Flows

ADF Data Controls, Connections and Data Sources

Task flows and pages use ADF data controls to fetch data from the backend. The data controls are backed by the ADF Connection mechanism. Identify the usages of data controls and the connections that back them in your framework application. These must be recreated on the WebCenter Portal Server by using Enterprise Manager or the appropriate WLST command. Your business logic may depend on data sources that are registered with WebLogic server. Even these must be recreated on the WebLogic server using the console, and they must be targeted to the managed server that is running WebCenter Portal.

More Information

» Monitoring and Configuring ADF Applications

Backing Bean and Business Logic

Your application will consist of managed beans that drive the page as is typical with a JSF/ADF-based application. Further, your application is likely to have a model layer consisting of various business logic, validations, and rules implemented in Java-based classes. To migrate this backend code, it is necessary to package them into .jar files. Look for potential opportunities to refactor the business logic in a backing bean into the model layer for easier maintenance and modularity. Backing beans should facilitate only the front-end and the business logic backing it. Consider different .jar files for backing beans and business logic. If you refactored some UI interactions into a task flow, ensure that you co-locate the backing bean and business logic into the same ADF library. You may want to create one JDeveloper application with two projects to manage this code: one to contain the backing bean code and another to contain the business logic.

This may be the toughest part of the migration task.

Once the Java code is moved to potentially several .jar files, add the jar files to the `extend.spaces.webapp` shared library. We will discuss the ways to add this shared library to WebCenter Portal in the Extending Oracle WebCenter Portal server section.

More Information

» Using a Managed Bean in a Fusion Web Application

» Understanding the Fusion Page Lifecycle

Declarative Components & Custom Tag Libraries

By far declarative components and custom tag libraries are the easiest to migrate because they are coded in a way that's already modular. You should add the .jar files to the `extend.spaces.webapp` shared library. We will discuss the ways to add this shared library to WebCenter Portal at the end of this section.

JEE Filters, Servlets, and Faces Configuration

When an application based on a WebCenter Portal Framework Application is created, it contains the following filters and servlets:

Name	Class
JpsFilter	oracle.security.jps.ee.http.JpsFilter
trinidad	org.apache.myfaces.trinidad.webapp.TrinidadFilter
ADFLibraryFilter	oracle.adf.library.webapp.LibraryFilter
adfBindings	oracle.adf.model.servlet.ADFBindingFilter
WebCenterEventDispatcherFilter	oracle.webcenter.framework.events.dispatcher.EventDispatcherFilter
adfServletFilter	oracle.adf.share.http.ServletADFFilter
ADFPortletFilter	oracle.portlet.client.adapter.adf.ADFPortletFilter
lifecycleLockFilter	oracle.webcenter.lifecycle.filter.LifecycleLockFilter
sitemapFacesContext	oracle.webcenter.jaxrs.services.sitestructure.faces.context.FacesContextFilter
webCenterLoginCountFilter	oracle.webcenter.framework.service.WebCenterLoginCountFilter
restFacesContext	oracle.webcenter.jaxrs.services.sitestructure.faces.context.FacesContextFilter

Figure 11. List of all the out-of-the-box filters in a WebCenter Portal Framework Application

Name	Type	
Faces Servlet	Servlet Class	javax.faces.webapp.FacesServlet
resources	Servlet Class	org.apache.myfaces.trinidad.webapp.ResourceServlet
BIGRAPHSERVLET	Servlet Class	oracle.adfinternal.view.faces.bi.renderkit.graph.GraphServlet
BIGAUGESERVLET	Servlet Class	oracle.adfinternal.view.faces.bi.renderkit.gauge.GaugeServlet
MapProxyServlet	Servlet Class	oracle.adfinternal.view.faces.bi.renderkit.geoMap.servlet.MapProxyServlet
GatewayServlet	Servlet Class	oracle.adfinternal.view.faces.bi.renderkit.graph.FlashBridgeServlet
adflibResources	Servlet Class	oracle.adf.library.webapp.ResourceServlet
adfAuthentication	Servlet Class	oracle.adf.share.security.authentication.AuthenticationServlet
ojsp	Servlet Class	oracle.jsp.runtimev2.JspServlet
ProfilePhotoServlet	Servlet Class	oracle.webcenter.peopleconnections.profile.internal.view.webapp.ProfilePhotoServlet
adfextapplogin	Servlet Class	oracle.adf.extapp.ExtAppLoginController
adfportlet	Servlet Class	oracle.adfinternal.view.faces.renderkit.html.portlet.ADFPortletServlet
resourceproxy	Servlet Class	oracle.portlet.client.resourceproxy.adf.ADFPortletResourceServlet
GetHandler	Servlet Class	oracle.webcenter.content.http.GetHandlerServlet
PortalAdminServlet	Servlet Class	oracle.webcenter.portalwebapp.servlet.PortalAdminServlet
PortalErrorServlet	Servlet Class	oracle.webcenter.portalwebapp.servlet.PortalErrorServlet
SiteMapServlet	Servlet Class	oracle.webcenter.portalwebapp.servlet.SiteMapServlet
JerseyWebApplication	Servlet Class	com.sun.jersey.spi.container.servlet.ServletContainer

Figure 12. List of all the out-of-the-box servlets in a WebCenter Portal Framework Application

The WebCenter Portal Server application comes configured with some of the inbuilt filters and servlets listed above. The functionalities provided by other inbuilt servlets and filters are provided by the application itself. Therefore, all the inbuilt servlets and filters configuration can be discarded while migrating to the Portal Server.

Custom-built Filters and Servlets

Your framework application may contain some custom built JEE filters and servlets. Consider revisiting the need for JEE artifacts in the Framework app and see how relevant they are on Portal. This consideration is specific to your Portal Framework Application and outside the scope of this document.

Consider addressing the purpose of the filters and servlets in the implementation of the backing bean or task flows. Alternatively, consider implementing the servlets as REST/SOAP based web services and use them in a task flow.

Faces Configuration

Out-of-the box, the `faces-config.xml` file has a custom view handler (`...handler.CustomViewHandler`), phase listener (`...view.SkinPhaseListener`), and a managed bean (`...preference.PortalPreferences`). You are not required to migrate all of these because the functions they provide are natively available on the Portal Server.

Extending Oracle WebCenter Portal server

The WebCenter Portal application can be extended through a shared library called `extend.spaces.webapp`. One can add artifacts, such as task flows, Java code, and other resources, to WebCenter Portal runtime by adding their libraries to this shared library. This is central to migrating custom-built components into WebCenter Portal. Consider bundling all of the artifacts mentioned above into this shared library and use them at runtime. Any changes to the artifacts involves redeploying the shared library alone. Therefore, you will have smoother product upgrade cycles

because your code remains isolated from the workings of the WebCenter Portal itself.

More Information

» Developing Task Flows, Data Controls, and Managed Beans for WebCenter Portal

» Development Lifecycle for Task Flows in Oracle WebCenter Portal

Migrating Portlets' Metadata

Your framework application may use Portlets on the pages. It takes two steps to migrate pages with portlets:

1. Create a connection to the Portlet Producer on the WebCenter Portal Server.
2. Migrate the portlet instance metadata from the framework application to the WebCenter Portal Server.

Creating Connection to a Portlet Producer

You can create a connection to portlet producer in multiple ways. Note the portlet producer connection details from the framework application and follow these steps to create the connections on the WebCenter Portal Server:

1. Use WebCenter Portal Administration.
2. Use Enterprise Manager.
3. Use WLST command.

Register a Portlet Producer

Navigate to the **Tools and Services** page in Portal Administration using the URL `http://host:port/webcenter/portal/builder/administration/tools`, and click **Portlet Producers**.

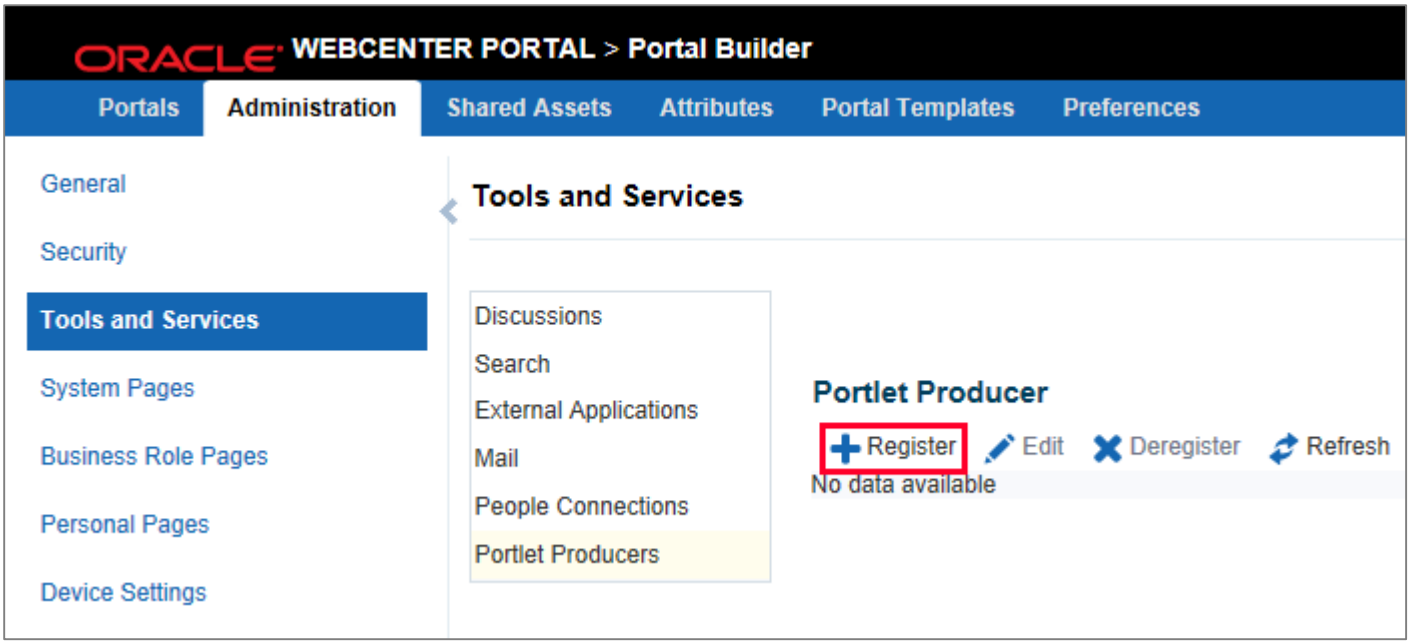

Figure 13. Register a portlet producer in Portal Administration

Register your portlet producer by choosing the Producer Type and providing the relevant information. Remember to test the information by clicking the Test button.

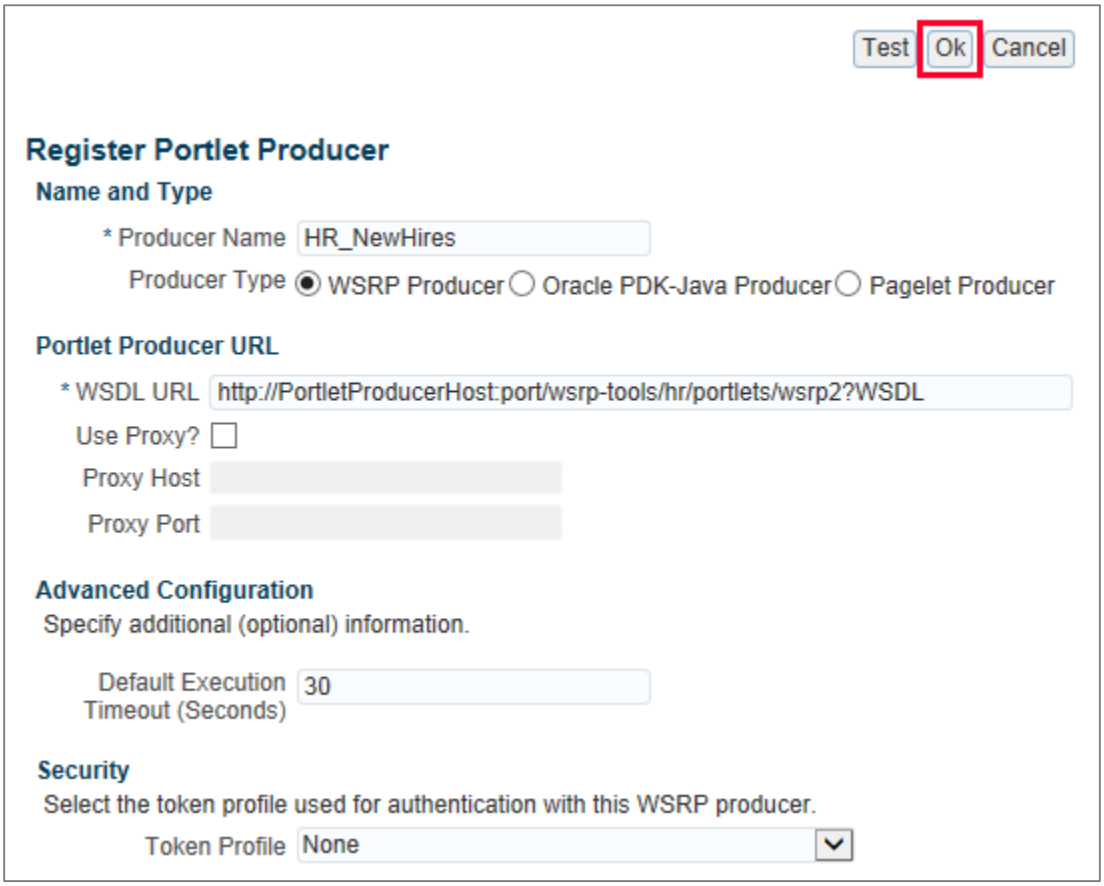

Figure 14. Provide information about a portlet producer

The portlet producer registration is complete. Repeat the steps to register all the portlet producers your Framework Application uses.

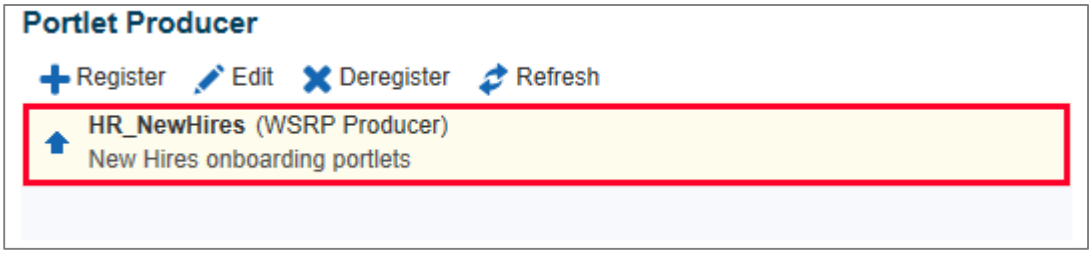

Figure 15. Portlet producer listed after successful registration

More Information

- Fusion Middleware Administering Oracle WebCenter Portal to register a Portlet Producer using Enterprise Manager
- Fusion Middleware WebLogic Scripting Tool Command Reference to register a Portlet Producer using WLST commands

Migrating Portlet Instance Metadata to WebCenter Portal Server

Every time a portlet is added to a page, its instance metadata is created and is associated with the application. It is important that these metadata are migrated to the Portal Server. Otherwise, the portlets on the migrated page will be broken. Portlets can be added to pages from within JDeveloper at design time or to the page at runtime using the Design-Time at Runtime (DT@RT) tool provided in Oracle Composer. To migrate portlet instance metadata, export them to a file using WLST commands—

```
$ cd $FMW_HOME/common/bin
$ ./wlst.sh
wls: > connect(...)
wls: > exportProducerMetadata(appName, fileName, [server, applicationVersion])
```

Where:

» `appName` refers to the name of the Framework Application.

» `filename` refers to the name of the file you want to export the portlet metadata into.

The rest of the parameters are optional. Refer to the documentation for more details.

Next, import the portlet instance metadata into the WebCenter Portal server with the following commands:

```
$ cd $FMW_HOME/common/bin
$ ./wlst.sh
wls: > connect(...)
wls: > importProducerMetadata(appName, fileName, [server, applicationVersion])
```

Where:

» `appName` refers to the name of the Framework Application.

» `filename` refers to the name of the file you want to export the portlet metadata into.

The rest of the parameters are optional. Refer to the documentation for more details.

This will move the portlet metadata from the managed server on which the Framework Application is running into the WebCenter Portal Server.

Exporting Portlet Metadata from JDeveloper

If you added portlets to the page only in JDeveloper and not at runtime using Oracle Composer, you can export the portlet instance metadata from within JDeveloper. Open the Framework Application in JDeveloper and use the 'Export Portlet Producers...' option in the Application menu to export the portlet metadata.

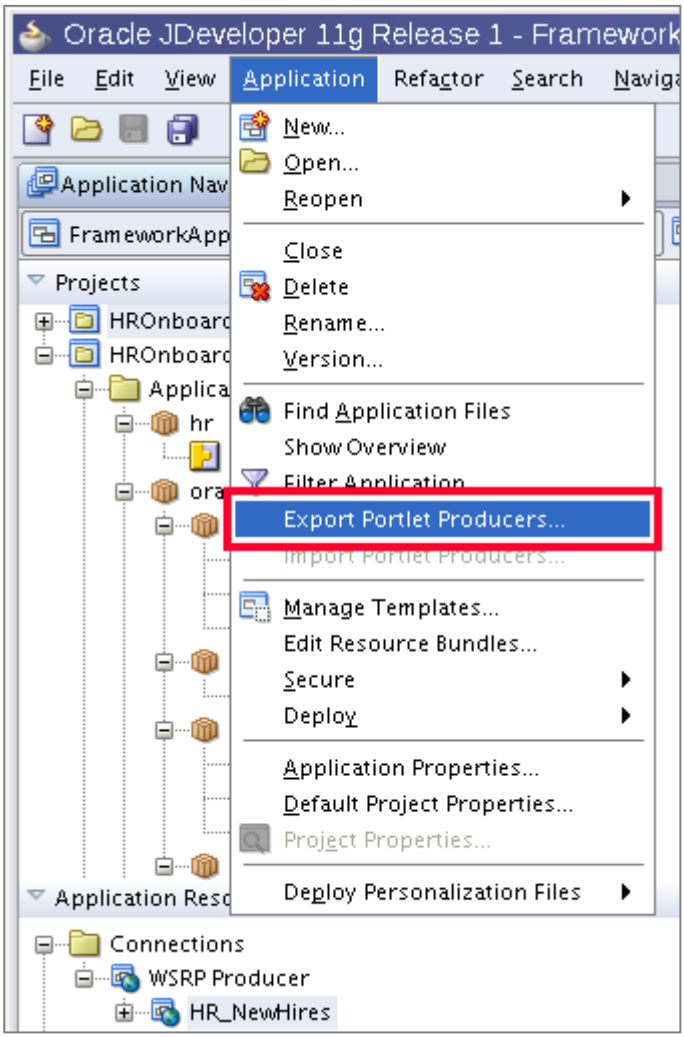

Figure 16. Export portlet metadata from a JDeveloper application

You will be prompted to enter a file name where JDeveloper will export the portlet metadata.

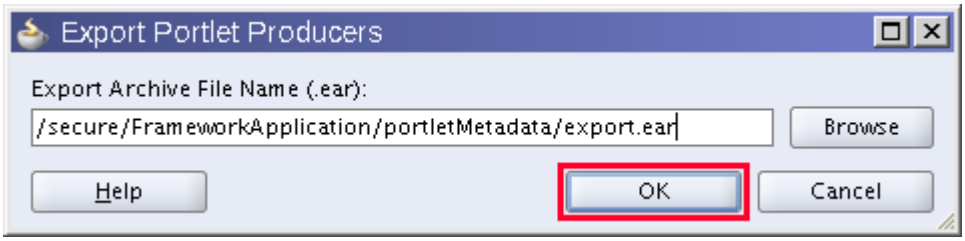

Figure 17. Provide a filename for the portlet metadata

You can use the `importProducerMetadata` WLST command to import the metadata to WebCenter Portal server.

Migrating Task Flow and Page Customisations

The framework application may have customizations applied to out-of-the-box task flows, such as for Favourites task flow or Document Library task flows. During the migration, you might want to carry forward the customizations to the portal server.

Note: Oracle WebCenter Portal supports customization at three levels (MDS Layers)—one, at the server instance level; two, at the individual portal level; three, at the user level. If your application has more layers than this, you should try and reduce the layers to the ones supported by WebCenter Portal.

In this section, you will learn how to map the MDS Layers in your Portal Framework Application to the ones supported by WebCenter Portal. All customizations to pages are scoped to a portal. However, you can scope the customizations done on an ADF task flow to any of the MDS layers supported by WebCenter Portal.

Table 2. List of supported MDS layers in WebCenter Portal

WebCenter Portal Customization Layer	**Details**
System	Customizations in this layer are applied to all the task flow usages in the portal server instance. The customisations are visible on all pages, and portals and to all users of the portal server. MDS Layer Name/Value: `site` / `webcenter`
Portal	Customizations in this layer are applied to all the task flow instances on all the pages of given portal. Customisations scoped to a portal are seen by all members of the portal and only when the portal is loaded. The customisations are not visible on any other portal or to the members of the portal when they are viewing other portals. MDS Layer Name/Value: `scope` / <`Internal Id/GUID` of a portal>
User	Customisations in this layer are applied to all the task flow instances on all the pages and portals for a given user. Other users will not see these customizations. MDS Layer Name/Value: `user` / <LDAP `GUID` of a user>

WebCenter Portal applies MDS customisations in the following order—System, Portal, User. That is, user-level customisations are applied on top of portal-level and both on top of system-level.

If your framework application has task flow customisations, consider scoping them to the portal you created to represent your application. Customisations will be visible only to the members of this portal if it is scoped to the *portal* layer. That way, the customisations will not be seen on other portals and by other users where it may not be relevant.

You may have multiple Portal Framework Applications to migrate to WebCenter Portal server. If you want these task flow customisations to be visible to users across all the portals, the customisations should be scoped to the *site* layer.

User customisations of task flows in the framework application are applied to all instances of task flows across all the pages and portals when they are migrated to portal server. It is not possible to scope user customisations to a given portal.

Migrating task flow customisations

You can export task flow customizations from the Framework Application using MDS' WLST command:

```
$ cd ${FMW_HOME}/common/bin/
$ ./wlst.sh
wls: > connect('user', 'password', 't3://host:port')
wls: > exportMetadata('frmk_app_name','managed_server', '/frmk_tf_cust',
docs='/oracle/webcenter/<task_flow_namespace>/**')
```

Where:

- `user`, `password` refers to the credentials of the WebLogic Server admin user.
- `host`, `port` refers to the host name and port of the WebLogic Server Admin server that manages the server where the framework application is installed.
- `frmk_app_name` refers to the name of your framework application.
- `managed_server` refers to the name of the managed server where your Portal Framework Application is deployed.
- `/frmk_tf_cust` refers to the location on the file system where you want the MDS WLST command to export the files into.
- `task_flow_namespace` refers to the namespace of the task flow's view/XML/view's page definition file.

Ex. The following is customization a fragment of *Activity Steam – Quick View* which is customised at MDS layer `site` and value `site` (loosely implying that this customization was done at the System level)—

`/oracle/webcenter/activitystreaming/view/jsf/fragments/`**`mdssys/cust/site/site`**`/activitySummaryCoreView.jsff.xml`

A list of all the customisable task flows available with Oracle WebCenter Portal is given in the More Information section. The list includes the namespace of the task flow definition file. This will be helpful in determining the namespace to use while exporting the customization fragments.

WebCenter Portal supports three layers—`site`, `scope` and `user`. Therefore, you must rename the MDS customization layer and value so that they continue to work once imported into WebCenter Portal server. The following table explains how the customization fragment can be scoped to portal server, portal, and to an individual user. Consider mapping any additional MDS customization layers to the ones supported by WebCenter Portal. If that is not possible, you should drop them altogether.

Table 3. Way to make WebCenter Portal identify task flow customization fragments

WebCenter Portal Customization Layer	Details
System	Rename the customization layer's value to `webcenter`. For this, you should rename the folder that goes by the layer value— `/oracle/webcenter/.../mdssys/cust`**`/site/webcenter`**`/activitySummaryCoreView.jsff.xml`

Portal	Rename the customization layer's value to the portal's internal ID (GUID). For this, you should rename the folder that goes by the layer value— `/oracle/webcenter/.../mdssys/cust/`**`scope/sc34364ff_57a6_47f7_a53a_d028d7181fdc`**`/activitySummaryCoreView.jsff.xml` The portal's GUID is displayed on the 'Overview' tab while administering it.
User	Rename the customization layer's value to the user's GUID. For this, you should rename the folder that goes by the layer value — `/oracle/webcenter/.../mdssys/cust/`**`user/joe.doe`**`/activitySummaryCoreView.jsff.xml` The user's GUID is available from your user store.

Once you rename the folders that contain the MDS customization fragments for the respective layer value, you are ready to import that into the WebCenter Portal Server. Import the customization into the WebCenter Portal Server for each of the task flow namespaces:

```
$ cd ${FMW_HOME}/common/bin/
$ ./wlst.sh
wls: > connect('user', 'password', 't3://host:port')
wls: > importMetadata('webcenter','WC_Spaces', '/frmk_tf_cust',
docs='/oracle/webcenter/<task_flow_namespace>/**')
```

Where:

- `user`, `password` refers to the credentials of the WebLogic Server admin user.
- `host`, `port` refers to the host name and port of the WebLogic Server Admin server where manages WebCenter Portal Server.
- `webcenter` refers to the name of WebCenter Portal application.
- `WC_Spaces` refers to the name of WebCenter Portal managed server.
- `/frmk_tf_cust` refers to the location on the file system where you exported and fixed the framework application's task flow customisations in the previous step.
- `task_flow_namespace` refers to the namespace of the task flow's view/XML/view's page definition file.

Based on the MDS customization layer mapping, the customisations should show up immediately.

Migrating page customisations

The mechanism to migrate pages and their page definitions is covered in the Migrating Pages section. Specifically, the `exportMetadata` WLST command exports customisations of pages and their page definitions. Your framework application can have multiple MDS layers. When multiple MDS layers are present, you must:

1. Map those layers to the ones supported by WebCenter Portal.
2. Export the pages and their page definitions from the framework application.
3. Rename the folder that represents an MDS layer as explained in Table 3.
4. Rename the file of the exported pages and page definitions to match the file name of the pages and page definitions created in the Portal Server. This is important because this is how MDS maps the pages with their customization fragments. Otherwise, the customisations will not take effect.

5. Import the pages and their page definitions into the WebCenter Portal server.

As with the `exportMetadata` WLST command, the `importMetadata` command imports the customization fragments of pages and their page definitions.

More Information

- Metadata Services (MDS) Custom WLST Commands
- Catalogue of Customizable Oracle WebCenter Portal Tools and Services Task Flows

Conclusion

This white paper described how to migrate the most important aspects of your Portal Framework application—pages, custom components, and Java code—to WebCenter Portal server. In subsequent white papers, we'll describe how to migrate portal assets, services' data, and the security model.

Oracle Corporation, World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065, USA

Worldwide Inquiries
Phone: +1.650.506.7000
Fax: +1.650.506.7200

ORACLE

CONNECT WITH US

blogs.oracle.com/oracle
facebook.com/oracle
twitter.com/oracle
oracle.com

Hardware and Software, Engineered to Work Together

Migrating Oracle WebCenter Portal Framework Application to Oracle WebCenter Portal Server
June 2015
Author: Sripathy Rao
Contributing Authors: Robin Fisher, Yannick Ongena, Paz Periasamy