

PREVIEW

Chapter 11

Runtime Customization

Please Note: All prepublication sample material is for review purposes only.
Content may change without prior notification.

If you have ever been part of building and deploying enterprise applications in the past, you may have faced one of the typical challenges: business users often tend to come forward with requirements when it's too late in the game; the application has either been rolled out or entered a state from which there is no return. Very often all they want is a tiny little functionality change: a slightly different page layout here, a simple announcement there, maybe content showing up on the page in a different order. To accommodate these types of business requirements most of the time you have to implement the changes in the development environment and after going through thorough testing and staging, you have to re-deploy your application over again. Not only can this be a very time consuming, and at times complex process, you also have to bring down your systems, often compromising your company's or customer's core business.

Wouldn't it be nice if instead, you could perform certain changes to your applications without going through the so unpopular testing and re-deployment process and you as an administrator could make these changes from within the deployed application? Or even better: if you as a developer had the means to design your applications in a way that empowers your business users to perform some or all of these tasks themselves simply by interacting with the running application?

If you have used portals before, it's no news to you that the above outlined is a core functionality in the portal space and it has been maturing for over a decade now. While the actual implementation may slightly differ, the following tasks can be performed in most of the enterprise portal products available in the market today:

- Edit content in the pages
- Add new and delete unneeded components from pages
- Re-arrange the order of components on the pages
- Re-arrange the layout of pages
- Contextually wire components on pages
- Edit component and page properties
- Create and delete pages at runtime

- Control page permissions at runtime

The promise of Oracle WebCenter is that you as an application developer can provide the above capabilities to your application administrators or business users at runtime.

In the not too distant past you had to weigh what was more important to you: building a transactional Java EE or JSF application, or providing the above listed portal capabilities. If you chose the former, you had limited or no portal capabilities. The latter implied that you couldn't take advantage of many features development frameworks provide.

With the WebCenter Framework being an integral part of Oracle's Fusion Middleware development platform, you don't have to make this choice any more. You can build your transactional composite SOA applications and pull in portal capabilities on an as needed basis. This chapter unfolds how you can add all these functionalities to your ADF applications.

Runtime Customization Concepts

Before diving into the details of building customizable applications, let's discuss some of the most important concepts and building blocks that we'll use.

Runtime customization is an idea that portals have introduced to the Internet world. The way it's implemented may well be very different, but what the portal sub-culture means by runtime customization is very much the same: you can use a simple Web browser to perform fundamental changes to an already deployed application; you can create new pages, define and change the page layout, add new content and components to the pages by selecting them from a repository.

Let's take a quick look at the key players in the field of runtime customization in Oracle WebCenter:

- **Oracle Composer:** the user interface that the WebCenter Framework provides administrators and business users to perform the different types of runtime customizations. Oracle Composer provides means to actions, such as taking a page into Edit mode, changing the layout of your pages, or moving components around on the page using drag and drop. The changes you make to your pages are persisted by the Metadata Services (MDS).

- **Metadata Services (MDS):** the infrastructure responsible for storing and managing customization data. MDS provides a unified architecture for defining and using metadata in an extensible manner. MDS stores metadata in XML format. When you customize a page, your changes, often referred to as the delta, is stored by MDS in an XML representation. When a request is received by MDS, it takes the base XML document, applies all the changes (or deltas) on top of that, and returns the result to the requester. MDS supports layered customization. For example, the administrator can add an announcement component on the front page of the application, group managers can add components to pages relevant to their team, and end users can personalize their pages and components, visible only to themselves.
- **Resource Catalog:** a read-only repository of objects that can be dropped onto pages. Examples for resources: portlets, task flows, JSF view components, and documents. It's important to note that the same Resource Catalog infrastructure is used by business users for runtime customization as for JSF developers for building applications using JDeveloper.

The three key players of runtime customization, Oracle Composer, the Resource Catalog, and MDS are very tightly integrated. In this chapter you learn how to build customizable applications by leveraging these three building blocks. If you are interested in details how MDS work and how ADF applications can take advantage of the capabilities MDS provides, refer to Chapter 13: Metadata Services and Chapter 17: MDS and WebCenter.

The Oracle Composer Toolbox

To familiarize ourselves with the tool set provided by Composer, let's build a simple application that contains the Composer components. By the end of this section you will know how to build an application that features the following capabilities:

- End users can add components to the page to their liking and change the height of the portlets and task flows by dragging the bottom right corner of these components. These changes are persisted to MDS as end user personalization.
- Users with edit privilege can take the page into edit mode and do the following:
 - Change the page layout, for example from two-columns to three-columns.

- Browse the Resource Catalog and drop new resources onto the page.
- Remove resources from the page.
- Edit resource and page properties.
- Wire components contextually to enable inter-component communication.

The Oracle Composer capabilities are surfaced in JDeveloper as JSF View Components. As shown in Figure 1, the WebCenter Framework provides eight JSF view components to allow you to build customizable pages.

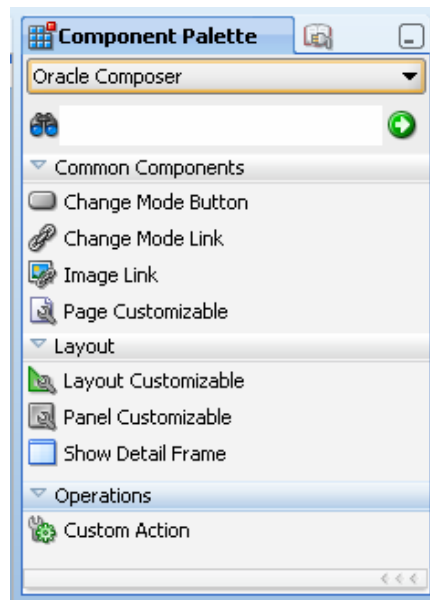


Figure 1: Oracle Composer on the Component Palette

- **Page Customizable:** this component defines the area of the page that is editable at runtime. When adding the Page Customizable component to the page, a Page Editor Panel is added as well as a facet. The Page Editor Panel provides the user interface to perform runtime customization operations, such as page and component property editing, contextual wiring, and showing the Resource Catalog.
- **Change Mode Button** and **Change Mode Link:** these components allow users with page customization permissions to switch the page from View mode to Edit mode.

- **Layout Customizable:** this component allows privileged business users to select from a set of predefined layouts, such as two-column or three-column, and apply it to the page.
- **Panel Customizable:** this component is a valid drop-target; portlets, Show Detail Frames, task flows, and JSF view components can be dropped into the Panel Customizable component from the Resource Catalog, as well as existing components on the page can be moved into them. In addition, all components within the Panel Customizable can be selected and edited at run time.
- **Show Detail Frame:** this component renders a header or chrome along with an optional border around its children components. The Show Detail Frame and its children can be dragged and dropped on the page from and to Panel Customizable parent components. The Show Detail Frame chrome provides the UI for dragging the component. In addition, the Show Detail Frame provides collapse/expand capabilities. It can also surface the custom actions of its child task flow.

The Show Detail Frame can contain only one child component; that is it only renders the first of its children. If you would like to surround multiple components with a Show Detail Frame, first lay them out in a grouping component, like Panel Group Layout, which can then be dropped inside a Show Detail Frame.

Note: The portlet view tag is very similar to the Show Detail Frame (in fact, it's a subclass of it), their behavior is very similar in a number of aspects: you can drag and drop and collapse/expand portlets, and portlets can present custom actions in their action menus as well.

- **Custom Action:** this component allows task flows to define task flow specific actions that are used to trigger navigational flow in the task flow. Custom actions are automatically exposed on the UI in the action menu of a Show Detail Frame component when the Show Detail Frame is the parent of the task flow.
- **Image Link:** this component can be used to include an HTML link as an image in the page.

Building a Customizable Application

Let's create a new application based on the WebCenter application template. For simplicity's sake we don't secure this application. When running the application from JDeveloper, you can perform all the tasks an administrator can do. You can test most of the functionality using a non-secure application. If you secure your application you can restrict the various tasks to users and roles with the right privileges.

If you don't have any application open in your Application Navigator, you can click the New Application... place holder. Alternatively, you can invoke the New Gallery by selecting the File > New... menu option. In the New Application dialog as well as in the New Gallery be sure to select WebCenter Application as your application template.

Let's call our application Customizable Application; you can leave all other application properties unchanged.

Let's create a new page, called customizablePage.jspx. Since MDS mandates us to use the XML representation of the page, select the Create as XML Document (*.jspx) option.

Note: The Create as XML Document (*.jspx) option and the extension of the name of your page you are about to create are synchronized to make the developer's life easier. When you check the option, the file extension changes to *.jspx, and vice versa, when you specify your file extension as *.jspx, the check box is checked automatically.

Also, to make sure that the contents of our page fills the entire browser page, select the Quick Start Layout option under Initial Page Layout and Content, as shown in Figure 2. If you plan to build a slightly more complex page, click on the Browse... button to choose from a variety of one, two, or three column page layouts. To keep this sample simple, we use the One Column (Stretched) layout option.

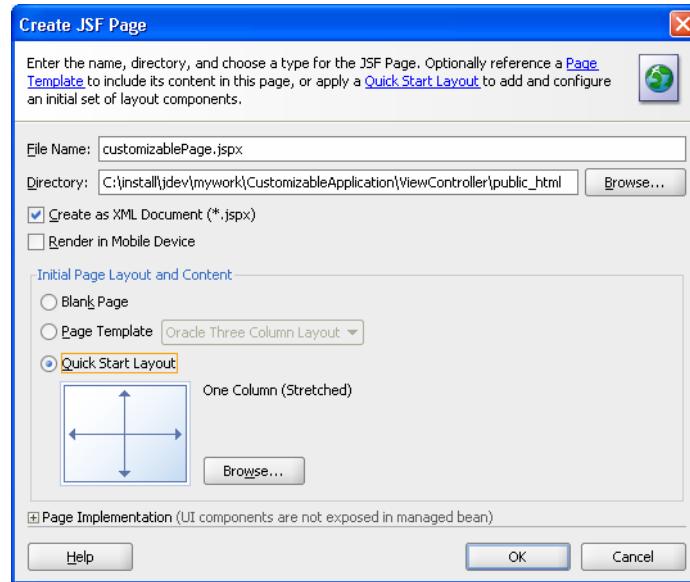


Figure 2: Create a New JSF Page

On the Component Palette select Oracle Composer from the page selector drop-down list and to make the page customizable, drop a Page Customizable component onto the page. Figure 3 demonstrates that the Page Customizable component pulls in a Panel Customizable component, as well as the Page Editor Panel in the edit facet. These additional components ensure that the desired customization capabilities can be achieved quickly and easily.

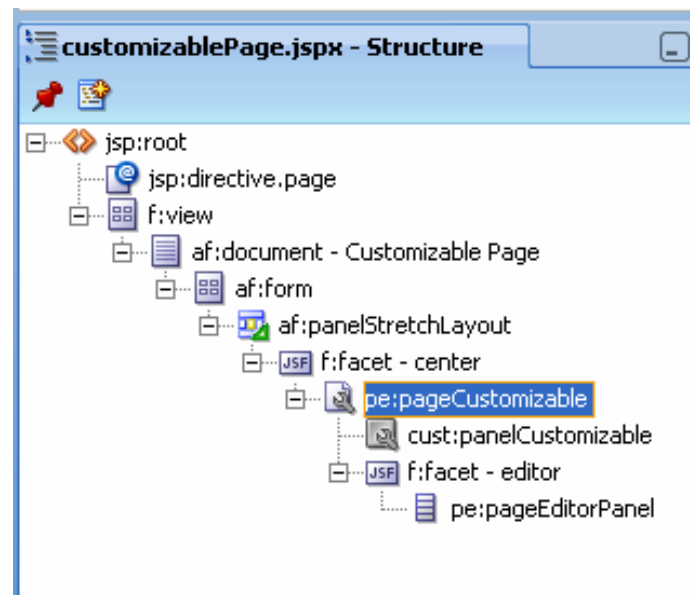


Figure 3: Page Customizable Component Dropped Onto a Page

In addition to the Page Customizable, we will need a button or link that will take the page into Edit mode. Let's add a Change Mode Button right above the Page Customizable component. When doing so, notice (Figure 4) that a Panel Group Layout is automatically added too.

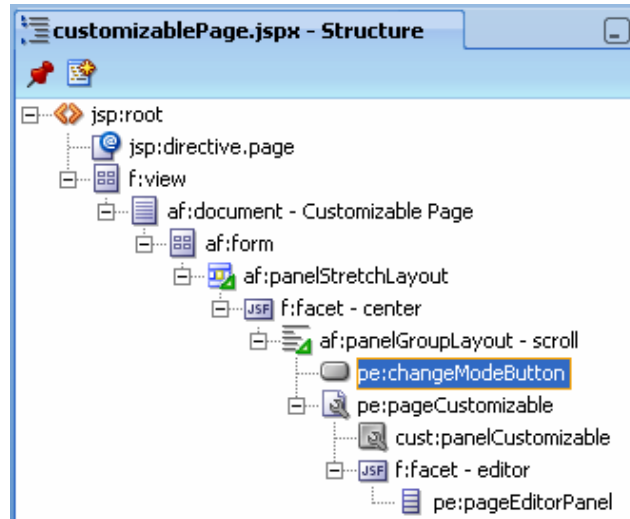


Figure 4: Customizable Page with Change Mode Button

After adding just these two components to the page, your application has a lot of customization capabilities. You can test at this point if you like. What you see after clicking the Edit button is presented in Figure 5.

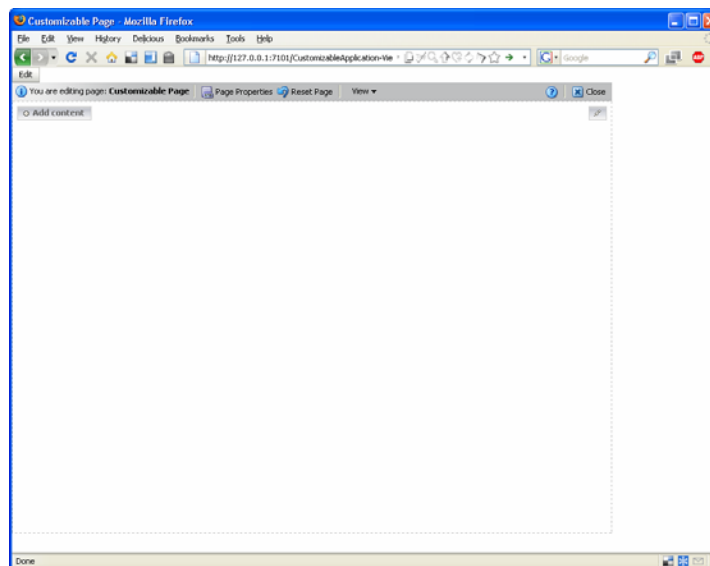


Figure 5: Running the Simplest Customizable Page

You may notice that the Page Customizable component doesn't stretch in the browser window. This is due to the behavior of the automatically added Panel Group Layout, which is a component that doesn't stretch its children.

There are several ways to make your Panel Group Layout stretch, we describe here one:

1. Enable the Top facet of the Panel Stretch Layout, as shown in Figure 6.
2. Move the Change Mode Button to the Top facet of the Panel Stretch Layout.
3. In the Source editor delete the Panel Group Layout.
4. Set the TopHeight attribute of the Panel Stretch Layout to 20px

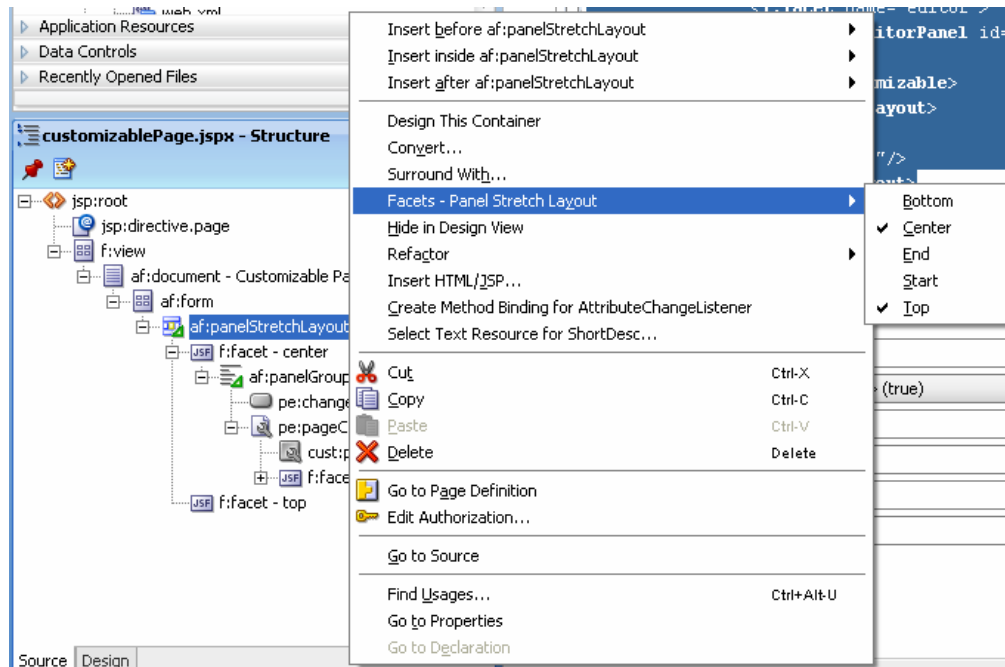


Figure 6: Enabling the Top Facet of the Panel Stretch Layout Component

Here is the source of the page after the above suggested modifications:

```
<?xml version='1.0' encoding='windows-1252'?>
<jsp:root xmlns:jsp="http://java.sun.com/JSP/Page" version="2.1"
  xmlns:f=http://java.sun.com/jsf/core
  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">
```

```
<jsp:directive.page contentType="text/html;charset=windows-1252"/>
<f:view>
  <af:document title="Customizable Page" id="d1">
    <af:form id="f1">
      <af:panelStretchLayout id="psl1" topHeight="20px">
        <f:facet name="center">
          <pe:pageCustomizable id="pageCustomizable1">
            <cust:panelCustomizable id="panelCustomizable1"
                                  layout="scroll"/>
          <f:facet name="editor">
            <pe:pageEditorPanel id="pep1"/>
          </f:facet>
        </pe:pageCustomizable>
      </f:facet>
      <f:facet name="top">
        <pe:changeModeButton id="cmb1"/>
      </f:facet>
    </af:panelStretchLayout>
  </af:form>
</af:document>
</f:view>
</jsp:root>
```

To make our example a little more interesting, let's add a few more components. First, we want to allow the page layout to be changed at run time. Let's replace the Panel Customizable (which was automatically added when we dropped the Page Customizable onto the page) with a Layout Customizable component. The Layout Customizable component allows us to change the page layout both at run time, as well as at design time. By default, the Layout Customizable component is visualized as a fairly small icon on the page. If you would like to display a text or label next to the icon, you can use the Text attribute to do so, for example: Select Layout.

Figure 7 shows how you can specify the seeded or default value for the desired page layout.

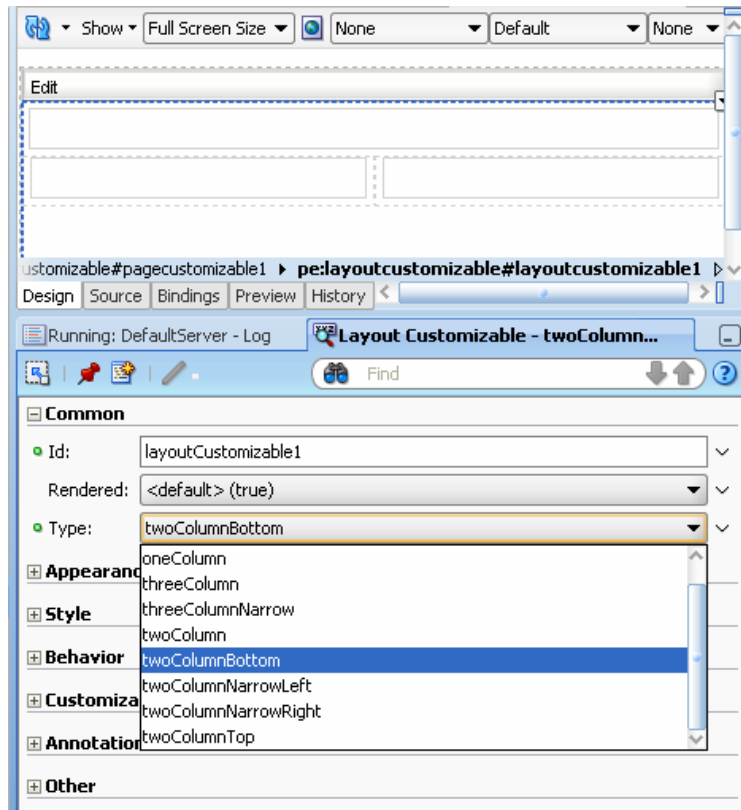


Figure 7: Specifying the Page Layout Type in JDeveloper

The Layout Customizable component manages three Panel Customizables, and lays them out in the page as specified by its Type attribute. The Layout Customizable offers eight content arrangement layout types, their names are pretty descriptive: oneColumn, twoColumn, twoColumnNarrowLeft, twoColumnNarrowRight, twoColumnTop, twoColumnBottom, threeColumn, and threeColumnNarrow. You can see the layout types in Figure 11. To restrict the offered layouts, you can provide a space separated list in the showTypes attribute of the Layout Customizable tag:

```
<pe:layoutCustomizable type="oneColumn" id="11"  
  showTypes="oneColumn twoColumn twoColumnNarrowLeft twoColumnNarrowRight">
```

As shown in Figure 8, one of the Panel Customizables is the child, and two others reside in the contentA and contentB facets.

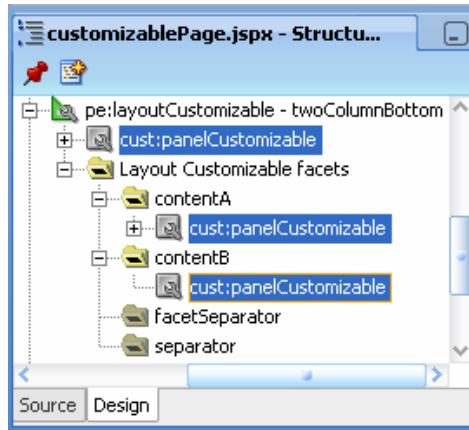


Figure 8: Panel Customizable Components Managed by Layout Customizable

Now, that we are done with the page customization settings, let's populate the page. We'll add a portlet and a calendar component in a Show Detail Frame.

When you run the page, your view should be similar to the one shown in Figure 9.

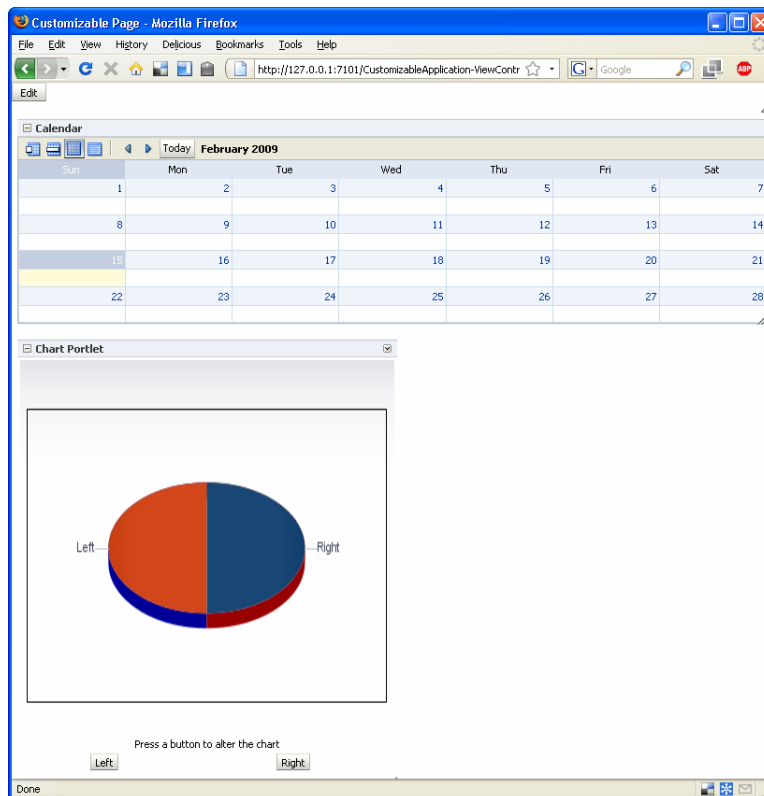


Figure 9: A Customizable Page with Components on it

Interacting with Oracle Composer

After having built a customizable application, let's discover what you can do with Composer at runtime.

The first thing you may notice is that both the portlet and the Show Detail Frame components provide a handle in their lower right corner that allows you to vertically resize them. Both of them support drag and drop as well, demonstrated in Figure 10.

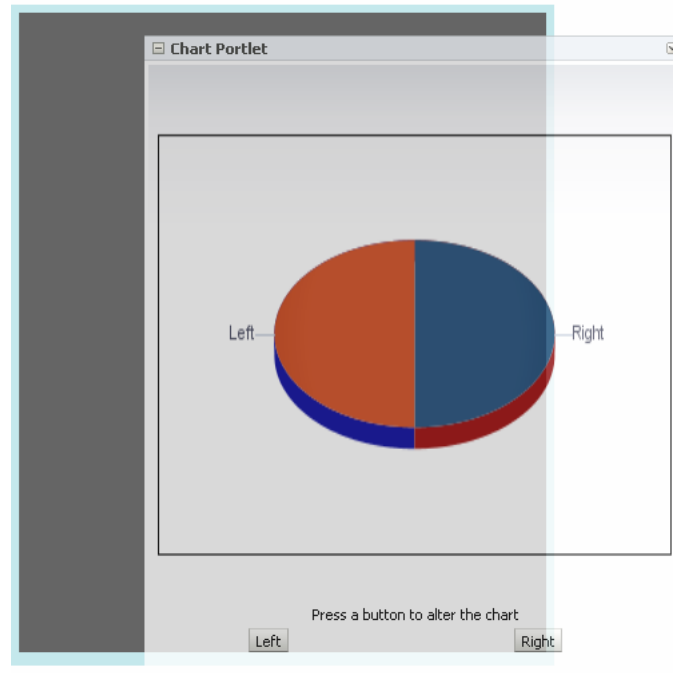


Figure 10: Dragging and Dropping Components on the Page

In the top right corner of the page click the little triangle icon. By doing so, a layout selector panel pops up, shown in Figure 11, allowing you to select the layout for your page.

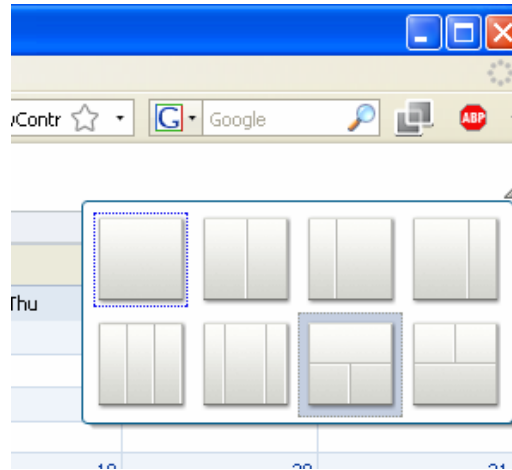


Figure 11: The Layout Selector Panel

To take the page into edit mode, click the Edit button. The information bar on the top of the editable section, shown in Figure 12, displays the name of the page that you are currently editing. By clicking the Page Properties button, you can change the name of the page as well as to add parameters to the page that can then drive and synchronize other components on the page. The Reset Page control removes all the runtime customization from MDS that was earlier applied to the page. By selecting the View Design or View Source options, you specify whether you want to exclusively see a WYSIWYG representation of the page, or you want to display a simplified tree view of the page as well. The latter may come handy when you have many nested components on the page and you want to select a specific one to set its properties.

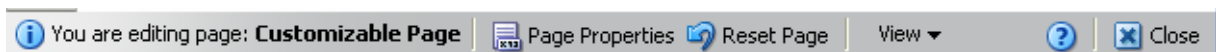


Figure 12: The Information Bar in Composer

To edit component properties, click on the Edit link, represented by a pencil, in the top right corner. The properties are shown in a floating pop-up panel and are organized in tabs. Most of the properties offer editor panels as well. Figure 13 gives you an idea what the component property editing panel looks like. This panel allows you to change the component title, or to contextually wire the component with other components on the page. Components can provide custom property editing panels as well.

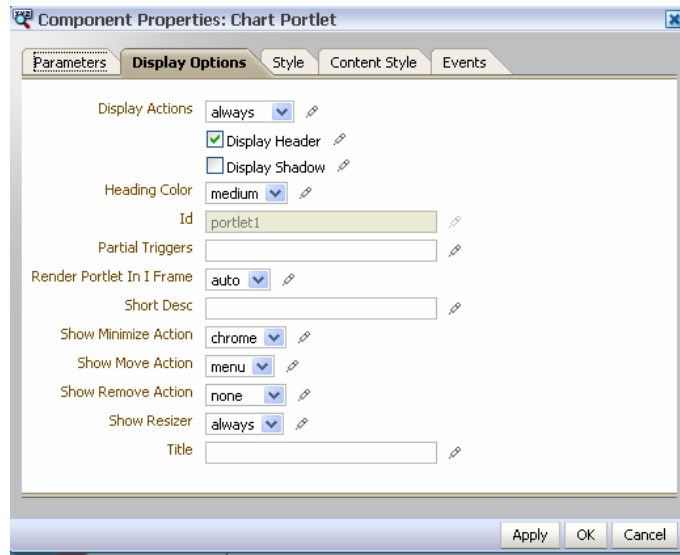


Figure 13: The Component Property Editing Panel

To add a new component to the page, select the Add content button in the Panel Customizable where you want to add the content to. This brings up the resource catalog pop up panel.

If you have registered portlet producers with your application before, the Portlet Producers folder shows up automatically in the Resource Catalog as shown in Figure 14. When drilling down in the Portlet Producers folder, all the registered portlet producers are listed. Selecting a specific portlet producer returns the list of portlets the producer offers. In our example we registered the Sample WSRP portlet producer with the application, therefore now we can add the Lottery Portlet, offered by this producer.

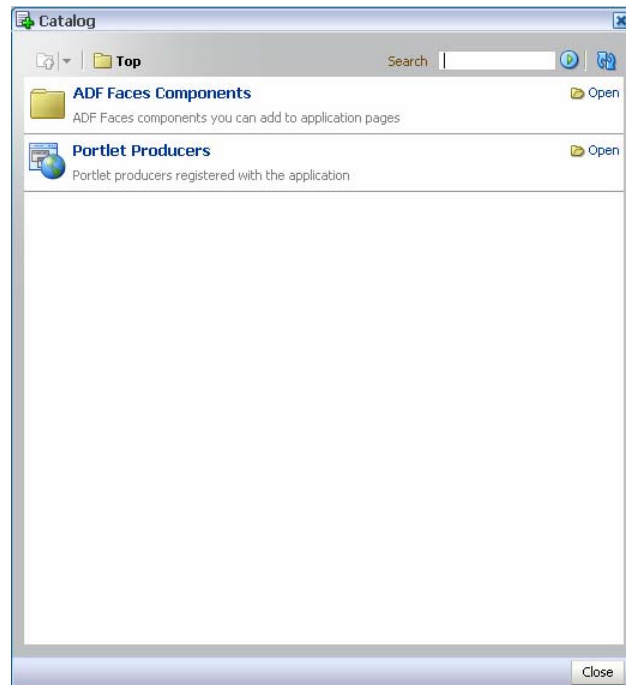


Figure 14: The Run Time Resource Catalog Viewer

When done, let's delete the Chart Portlet from the page by clicking on the Remove icon in the top right corner of the portlet.

Your view of the page should look similar to the one shown in Figure 15. To save some real estate, we collapsed the calendar component on the page.

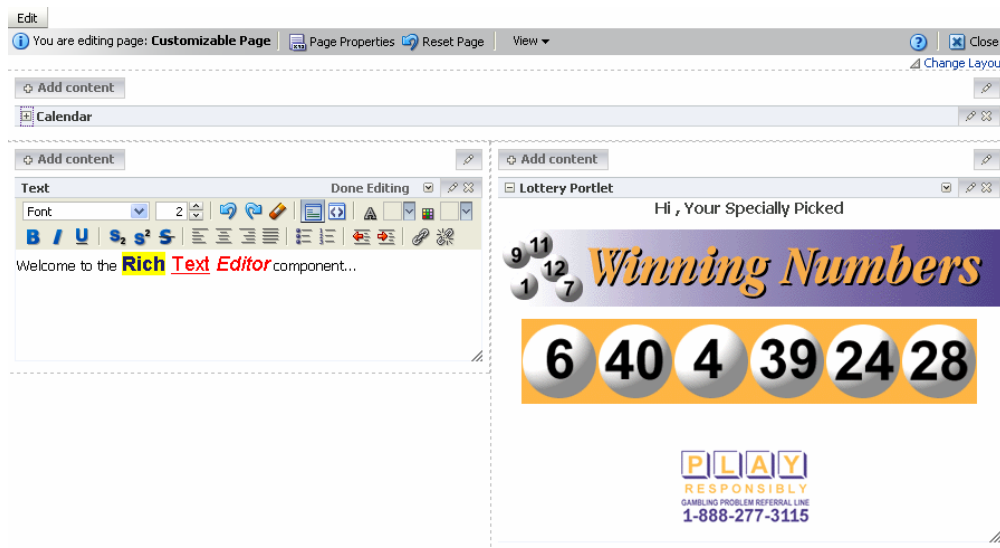


Figure 15: Page Modified at Run Tim

Cleaning Up Runtime Customizations

We have seen earlier, that Composer's Reset Page control on the information bar, demonstrated in Figure 12, removes all runtime customization applied to the page. The Build > Clean Runtime MDS Customizations menu option in JDeveloper does the same but for the entire application. It simply deletes the directory from your file system holding the MDS customizations. This menu option provides you an easy way to find the location of the file system repository MDS uses for your application at design time. By the way, this information is stored in `adf-config.xml` and can be changed for your application if need be.

Handling Concurrency

If multiple users are performing customization operations on the page at the same time, then all the independent changes get saved to MDS/.

If two users are customizing a page at the same customization layer, Composer's Information Bar displays a message letting users know that other users are editing the page as well. In this case changes that are committed last overwrite previous customizations.

Advanced Composer Configuration

Controlling Customization Options

The Show Detail Frame and Layout Customizable components provide you a lot of flexibility to control what you do and don't want to expose to your users. Both components can hide the edit action. In addition, the Show Detail Frame can hide all or some of the actions, including minimize, move, resize, edit, and remove. These options are controlled by the `ShowMoveAction`, `ShowRemoveAction`, `ShowResizer`, `ShowMinimizeAction`, and `ShowEditAction` properties.

Concurrent Page Editing Using Composer Sandbox

When two users are editing the page at the same time, the information bar, presented by the Page Customizable component, gives you a warning message about it. If you would like to

turn off this the warning, you can do so by setting the Page Customizable component's ShowMessage attribute to false.

One of the reasons Web applications are so successful is that they support the simultaneous access of the same application by a large number of users. When such applications offer features beyond read-only capabilities, dealing with concurrent access becomes essential.

When Composer is configured to use the file system repository, which is the default in a development environment, managing concurrent access is limited to displaying warnings to users. This can be elevated to the next level by configuring Composer to work against a database MDS repository. In this case you can enable Composer's so-called sandbox functionality. Sandbox in the Composer world is a temporary storage area to save run-time page customizations before they are either committed to the back end or canceled. If you don't use sandbox, your users' changes are automatically and immediately committed. If you are familiar with SQL, sandbox gives you the Composer version of savepoint-rollback-commit functionality.

Since the exact steps are documented in the Oracle WebCenter Guide here we only review the high level steps to configure sandbox for Composer:

1. To enable sandbox, you have to make sure that your MDS uses a database repository. If it doesn't, you have to re-configure and/or migrate your file system based MDS repository to a database repository.
2. The next step is editing your `adf-config.xml`, declaring for what metadata you want to enable sandboxing. Most likely you want to enable sandboxing for your `*.jspx` and page definition files.
3. In the `adf-config.xml` file turn sandboxing on.
4. Define a Composer-specific filter in your application's `web.xml`, before the ADF bindings filter. This will ensure that all requests are routed through the Composer-specific filter first.

At runtime, when you switch to Edit mode, the presence of a Save button on the header of the Composer panel is the most obvious indication that sandbox creation is enabled for the application.

After making the required changes in Edit mode, if you are satisfied with the changes you have made, then you can commit the changes by clicking Save on the Composer panel. Alternatively, you can click Close to close the Composer panel, and then you'll be prompted to save or cancel your changes.

Note: Oracle WebCenter Spaces takes advantage of the sandboxing feature, shown in Figure 16.

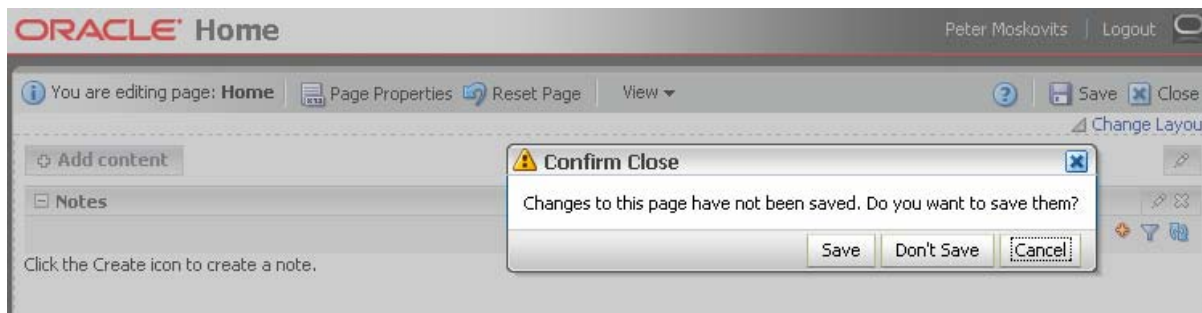


Figure 16: Composer Sandbox Used by WebCenter Spaces

Custom Actions

In addition to the common capabilities of the Show Detail Frame component, such as drag and drop or collapse and expand support, a less frequently used but quite powerful functionality it provides is the action menu in its top right corner. The flexible custom action framework allows you to plug in and surface task flow actions in this menu.

Let's take a look at the following example. Your calendaring task flow has a main view and a view showing print preview. The two views, demonstrated in Figure 17, are implemented as ADF task flow view activities. Now, in addition to (or instead of) providing navigation controls between the two view activities inside your task flow, you can also surface action menus in the Show Detail Frames surrounding the task flow, to provide the navigation between the two views.

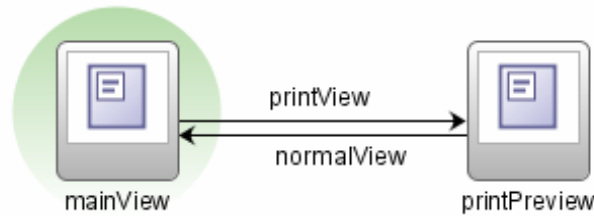


Figure 17: A Task Flow with Two View Activities

The first view activity provides two ways to navigate to the print preview. A traditional button on the bottom of the task flow, labeled as Print Preview, and a custom action link, demonstrated in the top right corner of Figure 18: Print Preview, with a printer icon next to the label.

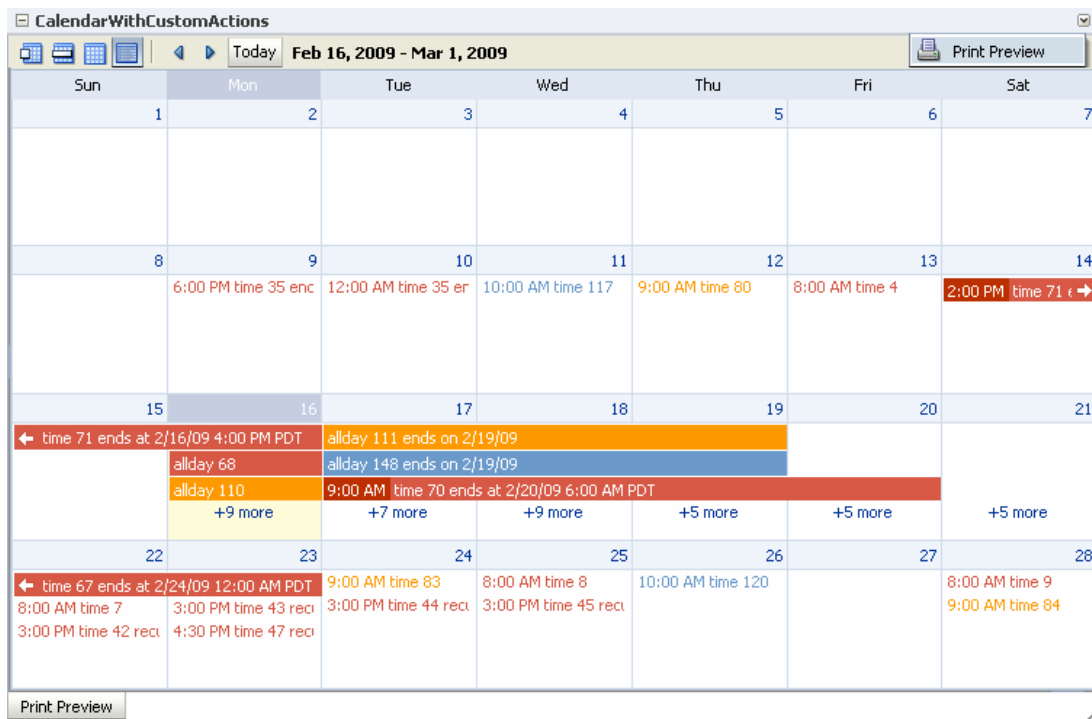


Figure 18: Task Flow View with Two Navigation Options to Print Preview

To navigate back from the printPreview view activity to mainView, the task flow provides a custom action, labeled as Back to Normal View, shown in Figure 19.

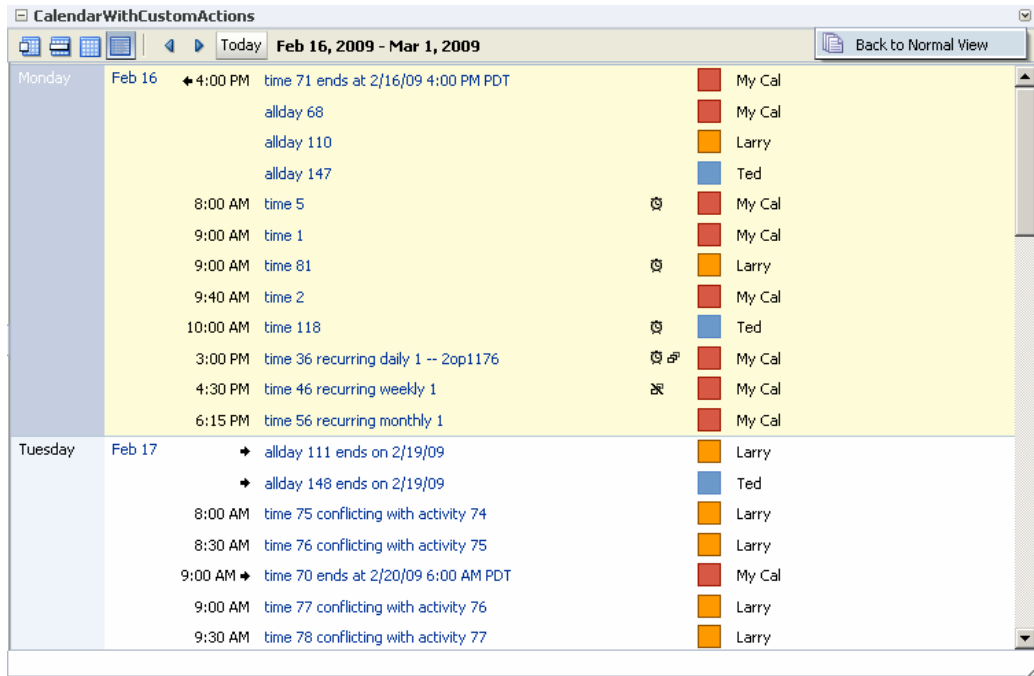


Figure 19: Task Flow Featuring a Custom Action Navigation Option

How can you create your own custom actions? Once you have the actions defined in your task flow definition file, shown in Figure 16 (called task-flow-definition.xml, by default), all you need to do is add one or more customAction tags to the Show Detail Frame holding your task flow. The two key things you can specify with the customAction tag: the label and the icon shown in the Show Detail Frame menu.

It's important to remember, that only those actions are displayed in the Show Detail Frame that are valid navigation actions from the currently displayed view activity in the task flow. This is why despite the two custom actions added to the Show Detail Frame, in our case only one is shown at any given time.

Here is the source code for your Show Detail Frame:

```
<cust:showDetailFrame text="CalendarWithCustomActions" id="sdf1">
  <af:region value="#{bindings.taskflowdefinition1.regionModel}"
    id="r1"/>
  <cust:customAction action="printView" text="Print Preview" id="ca1"
    icon="iconpath/print_ena.png"/>
  <cust:customAction action="normalView" text="Back to Normal View"
```

```
id="ca2" icon="iconpath/pages_qualifier.png"/>  
</cust:showDetailFrame>
```

On top of defining custom actions for individual Show Detail Frame components, you can also provide global custom actions, applied to all Show Detail Frames in the application.

A logical question at this point is: which of the globally defined actions show up in the menu of the Show Detail Frame? The behavior is the same as we saw before: only those actions are present in the menu that have a corresponding control flow in the currently displayed view activity of the task flow. If we are in normal mode, the print preview action is displayed, and vice versa, if we are in print preview mode, the normal view item is shown.

You can declare your global custom actions in the `adf-config.xml`:

```
<cust:adf-config-child>  
  <enableSecurity value="true" />  
  <customActions>  
    <cust:customAction action="printView" displayName="Print Preview"  
      id="gca1" icon="iconpath/print_ena.png"/>  
    <cust:customAction action="normalView" tooltip="Back to Normal View"  
      id="gca2" icon="iconpath/pages_qualifier.png"/>  
  </customActions>  
</cust:adf-config-child>
```

Creating Pages at Runtime Using the Page Service

Very much like making changes to page content and layout, creating pages at runtime is a typical functionality that every portal product provides. While the way how business users and developers interact with it is very different, the Page Service is most of the time discussed on the same page with Oracle Composer. They both provide powerful runtime application customization capabilities and both use MDS as their repository.

The page service lets the authorized business users of your application create new pages, delete pages, copy pages, and change page properties.

The Page - Create New Task Flow

The button, responsible for creating a new page, has been implemented as an ADF task flow, called Page - Create New. In addition to the Create Page button the task flow also provides a modal dialog window displaying the page styles that the user can choose from.

The Page - Create New task flow, just like any other WebCenter task flow, resides in the WebCenter Services Catalog, in the Resource Palette in JDeveloper. It accepts seven parameters. The easiest way to define values for the parameters is by navigating to the page definition file of the page containing the task flow, and opening the Parameters section in the Property Inspector. This is the list of the task flow parameters and what they control:

- **Scope name:** by assigning pages to scopes using the scope name parameter, you can group your pages
- **Outcome:** the outcome parameter defines the Java method that gets invoked after the page is created. For example, this allows you to provide visual feedback to the user after the page is created, or to navigate to the newly created page and take it into edit mode.
- **Page style file:** this parameter allows you to point to an xml file (called template.xml, by default) that describes the page styles that the user can pick from when creating a new page. Page styles are basically jsp pages, optionally accompanied by a page definition file, that is copied as the newly created page.
- **ADF template:** here you can define the ADF page template that you'd like your page to use. If your page style is based on an ADF template, you can configure the page creation task flow such that a different ADF template is used based on different criteria such as user or scopes.
- **User interface type:** this parameter allows you to specify whether you want your Page - Create New task flow to render as a link or a button. By default it renders as a button. To render as link, the parameter value has to be: `#{ 'link' }`
- **Icon:** the icon parameter allows you to specify a custom icon that will appear on your button or next to your link.
- **Label:** this parameter provides you control over the label that is rendered.

Figure 20 shows two visualizations of the Page - Create New task flow. The first one is the default, the second one with different icon and label.



Figure 20: Page - Create New Task Flow: Two Visualizations

When users click the button, a dialog is displayed allowing them to specify the name, color scheme, background color, and style for the new page, shown in Figure 21.

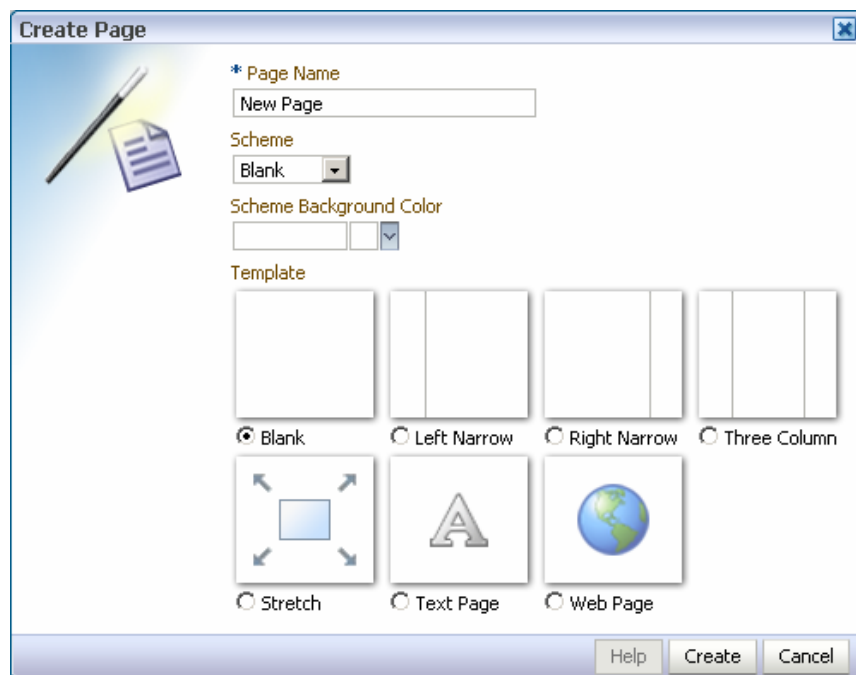


Figure 21: The Create Page Dialog

The Page Service Data Control

The Page Service data control, PageServiceDC, gives you access to information about your pages at runtime. It also lets you delete any of your pages at runtime.

In the following simple example you will see how to use the Create Page task flow to create pages at runtime, and what it takes to display the pages using the Page Service data control.

Create a new application using the WebCenter Application template. We create a page using a two column Quick Start Layout. In the WebCenter catalog locate the Page - Page Service task flow, and drop it onto the narrow column in your page (labeled as first).

Locate the PageServiceDC under Data Controls in your applications. Expand it, expand the getPageTree(String) node and drop the PageTreeNode onto the wide column of your page (labeled as second), shown in Figure 22.

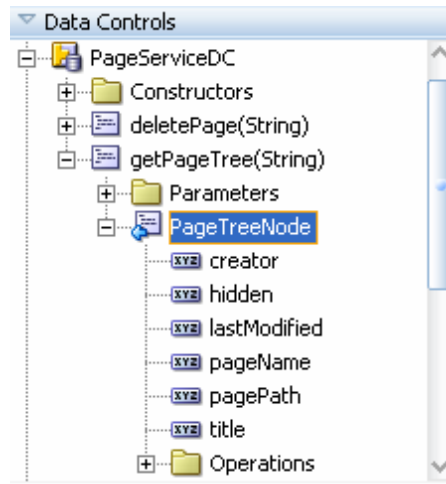


Figure 22: The Page Service Data Control

Visualize the data on the page as ADF Read Only table. Select the defaults, and click OK. When prompted for the scopeName parameter, you can leave it empty, and click OK again.

This table will display metadata about the pages in MDS. We will make one minor change to it: we add a Go Link that will allow us to navigate to the page displayed in the table.

Drop a Go Link into the first (title) column of the table, and delete the original Output Text rendering the title. Select the newly added Go Link, and change the following three properties:

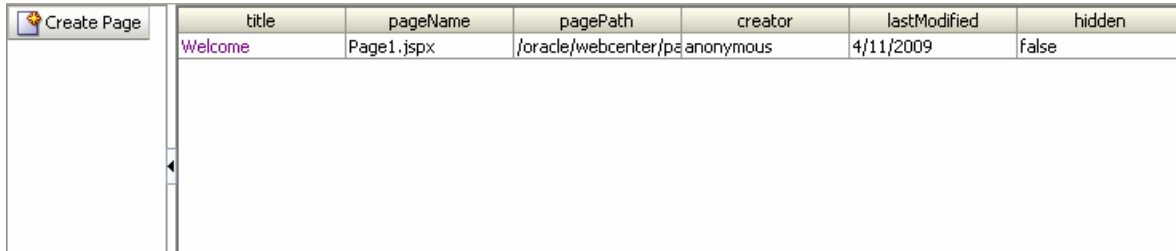
Text: `#{row.link}`

Destination: `/faces/#{row.pagePath}`

Target Frame: `_blank`

Let's run the page. Click the Create Page button, and specify a name, scheme, and page style you want to base your new page on.

Since our application is very simple, you'll have to perform a manual page refresh in the browser for the newly created page to show up in the table, as shown in Figure 23.



title	pageName	pagePath	creator	lastModified	hidden
Welcome	Page1.jspx	/oracle/webcenter/pa	anonymous	4/11/2009	false

Figure 23: Creating Pages at Runtime

When clicking on the link, you're taking to the newly created page, called Welcome. In our example we used the Text Page as the page style (or template), shown in Figure 24.

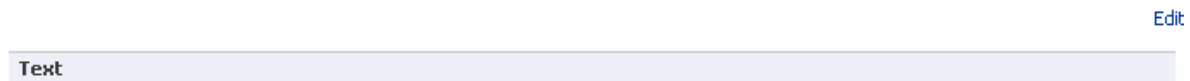


Figure 24: Page with Rich Text Editor, Created by the Page Service

The official Oracle documentation, the WebCenter Developer's Guide, available online on OTN, discusses the Page Service capabilities in a lot more detail. If runtime page creation and management is your bread and butter, the Developer's Guide, presenting nice examples, is a great resource to read.