ADF Code Corner

88. How to create nested page templates in Oracle JDeveloper 11g R2

twitter.com/adfcodecorner

Abstract:

Oracle JDeveloper 11*g* introduced page templates in ADF Faces for developers to build consistent page layouts. In JDeveloper 11.1.1.x, page templates cannot be nested or extended, which is a restriction that no longer exists for teplates built in Oracle JDeveloper 11g R2 (11.1.2). This blog article demonstrates how to create nested page templates and how to build templates that itself are based on templates.

Author: Frank Nimphius, Oracle Corporation
twitter.com/fnimphiu
02-AUG-2011

Oracle ADF Code Corner is a loose blog-style series of how-to documents that provide solutions to real world coding problems.

Disclaimer: All samples are provided as is with no guarantee for future upgrades or error correction. No support can be given through Oracle customer support.

Please post questions or report problems related to the samples in this series on the OTN forum for Oracle JDeveloper: http://forums.oracle.com/forums/forum.jspa?forumID=83

Introduction

The Oracle JDeveloper 11*g* R2 new features document mentions this new ADF Faces functionality in a small parapgraph:

Problem statement

"Previously, one was only able to utilize a single page template in their page (chosen during the creation of the page). Also, one was not able to include another page template while creating a page template definition."

The reason for joy

"That restriction has been removed and a new Template item is now available in the component palette which can be DnD'ed onto either a page template definition or a regular page."

How it works

" A dialog will appear prompting the user to choose a page template to point to. With this new feature, the ability to nest page templates is now possible."

This however does not full explain how developers can use this feature, and chapter 10, which explains how to build page templates in Oracle JDeveloper 11g,is not yet updated to contain this important piece of functionality.

First of all, nesting of templates always used to work in Oracle JDeveloper 11g at runtime. What has been missing so far is the design time for this and – most importantly – support. With Oracle JDeveloper 11g R2, page template inheritance and nesting is supported in design and runtime.

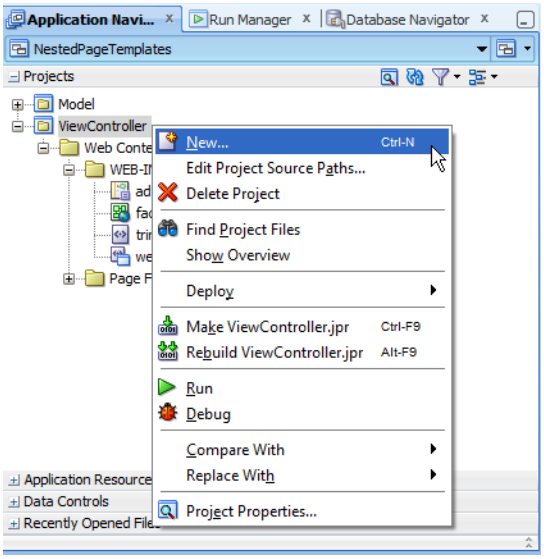
The advantage of extending a template, or nesting another within, is that developers now can break up page layout definitions in individual parts, like header, footer and body, which not only is more flexible but also increases the opportunities for template reuse.

With JDeveloper 11g R2, page designers should look at page templates, dynamic declarative components and JSF 2.0 composites, and skinning when building application layouts.

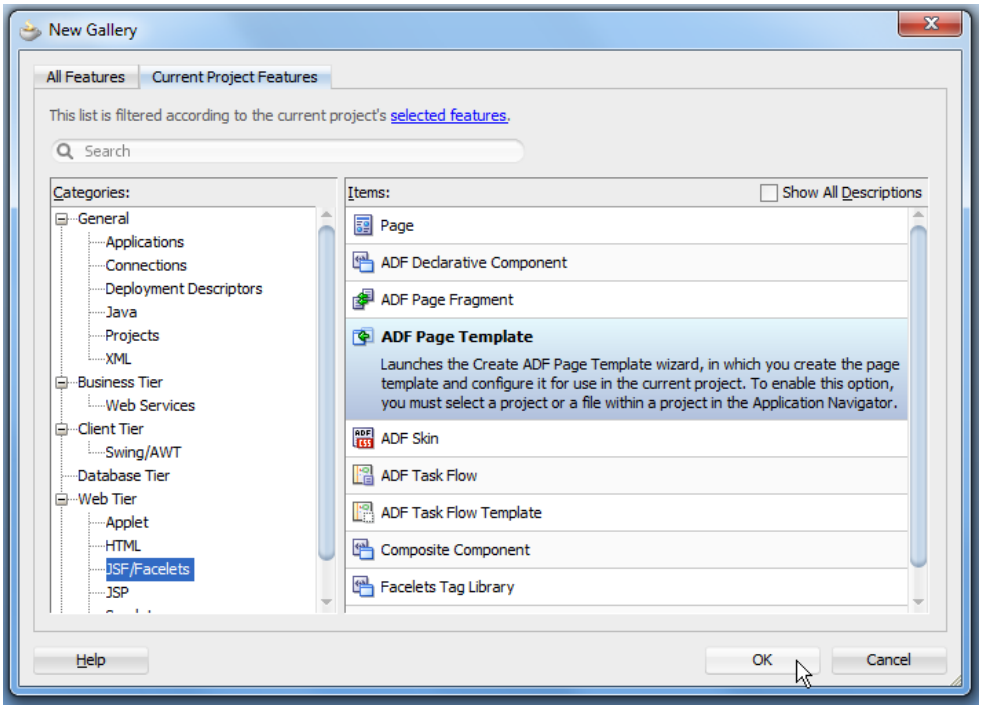
In this article, the page layout shown below is explained. The custom base template extends the Oracle three column page layout and adds two additional areas in the center. Another template defines three extra columns and is added as a nested template to the extended three column layout when building the page. In this article you also learn about a new feature for converting layout containers, which is a useful functionality changing an existing layout, for example to change the geometry management behavior.

Extending an existing Template

To build a new page template, start from the Oracle JDeveloper New Gallery that you open from the **File | New** menu option or the context menu shown below. If you have reuse in mind, then the new template is better created in its own JDeveloper project, which then makes it easier to JAR it up in an ADF Library and deploy it across applications.

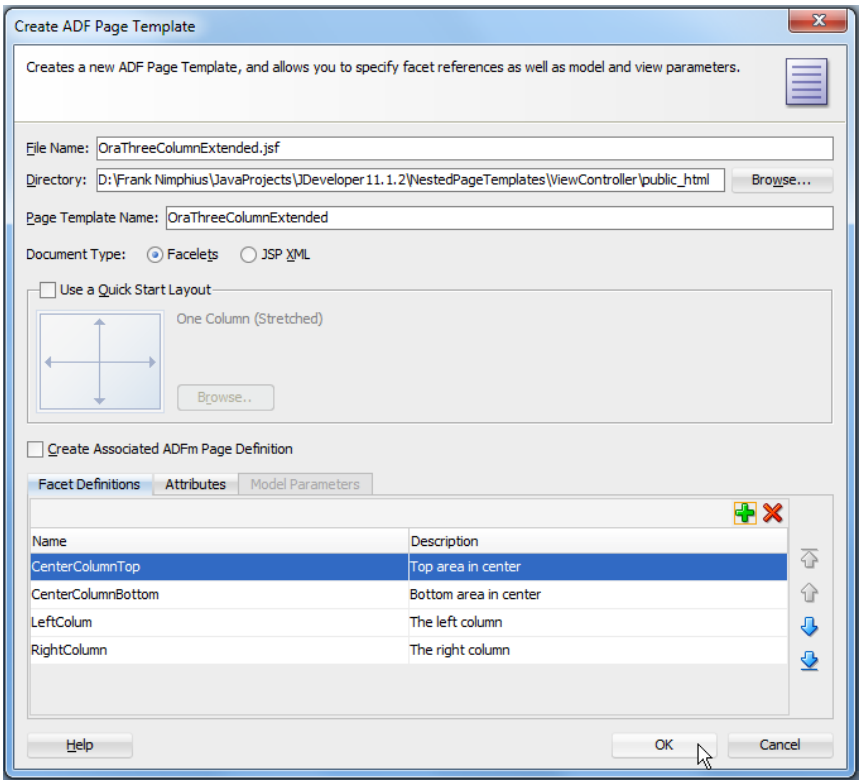


In the JSF section within the New Gallery, select the **ADF Page Template** option and press **Ok**.



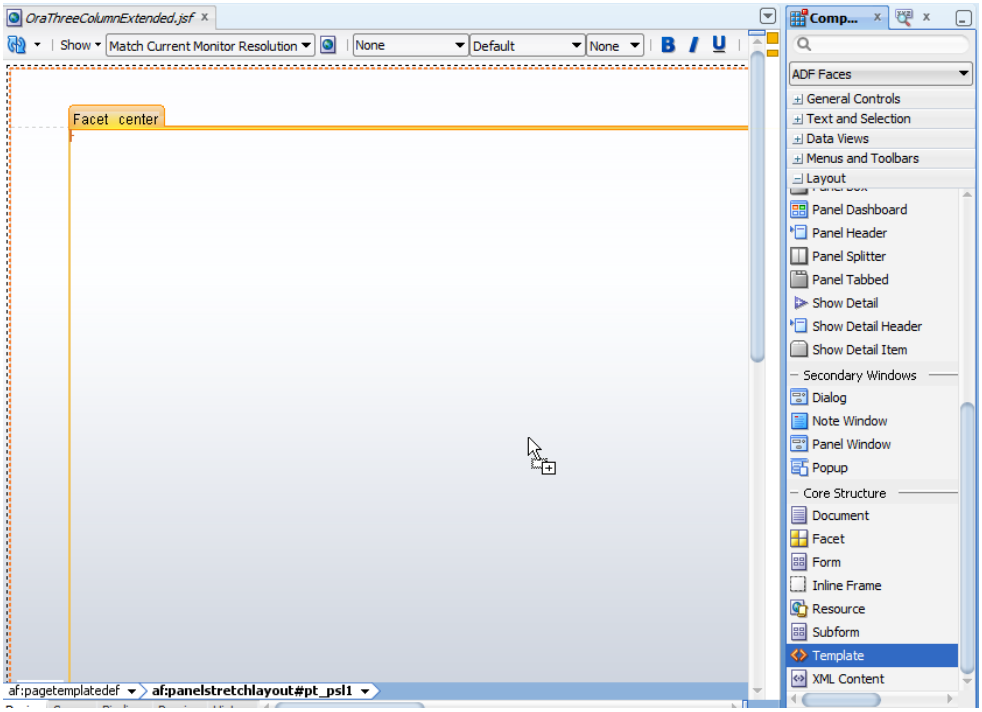
In the opened **Create ADF Page Template Create**, provide a name for the template and where in the directory structure it should be created in. If you build templates into an application, it makes sense to save template files in a sub-directory of the public_html directory.

To create a start layout for your custom page template, you can select the **Use Quick Start Layout** and choose from a list of pre-defined layouts. In this article, we start the new template with a blank page and add an `af:panelStretchLayout` component manually as the root component. It is recommended that page templates that are used as a base page template start with a component that stretches itself and also stretches its contained children.

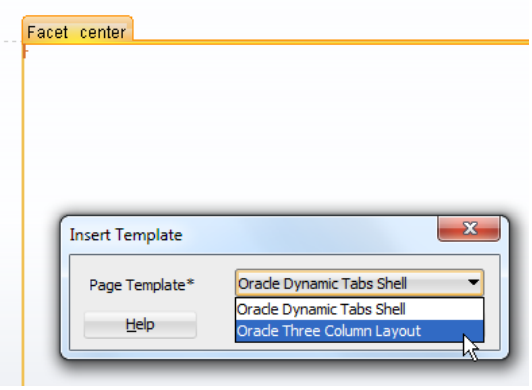


The **Facet Definitions** define areas within the template that application developers later add ADF Faces UI components, or even additional layout components.

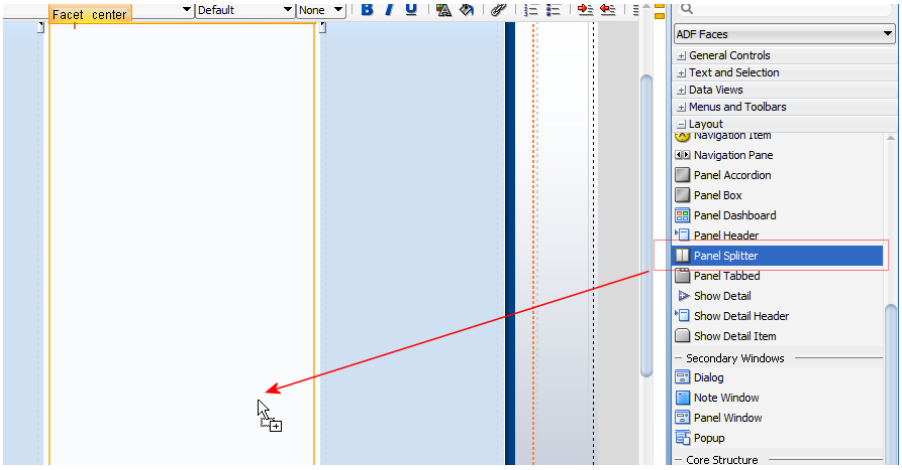
The page template shows in the Oracle JDeveloper visual editor after finishing the template creation by pressing **Ok**.



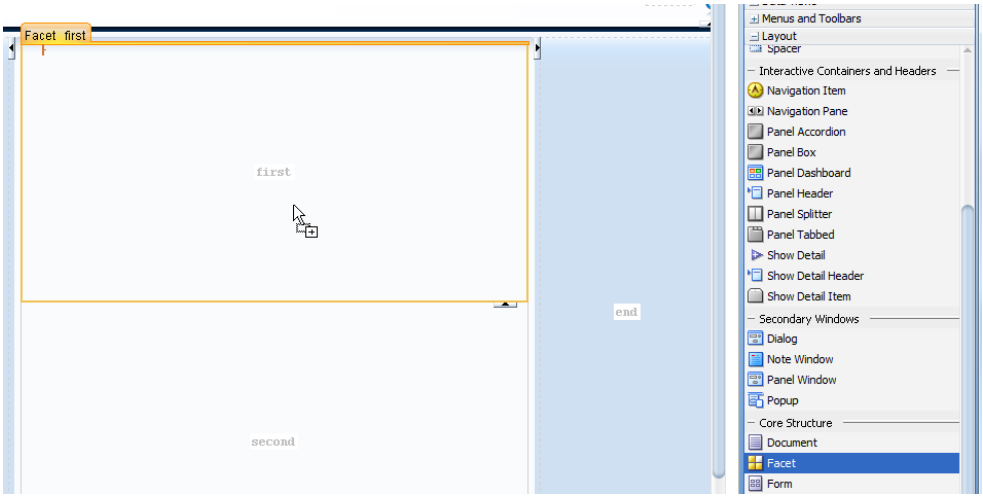
To make this template to extend an existing template, open the Component Palette and select the **Adf Faces | Layout | Template** option as shown in the image above.



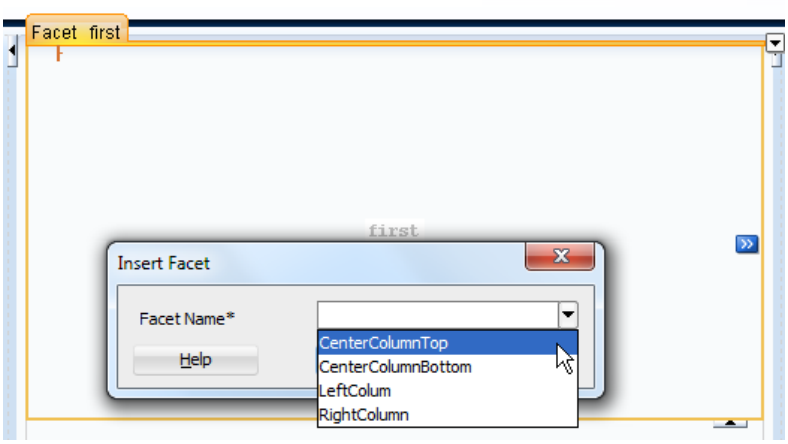
The **Insert Template** dialog that opens shows existing templates provided in an ADF Library or contained in the current project. For this article, select **Oracle Three Column Layout**.



To extend the existing three column layout, drag a panel splitter component from the JDeveloper component palette to the center column and use the Property Inspector to set its **layout** property to **vertical**.

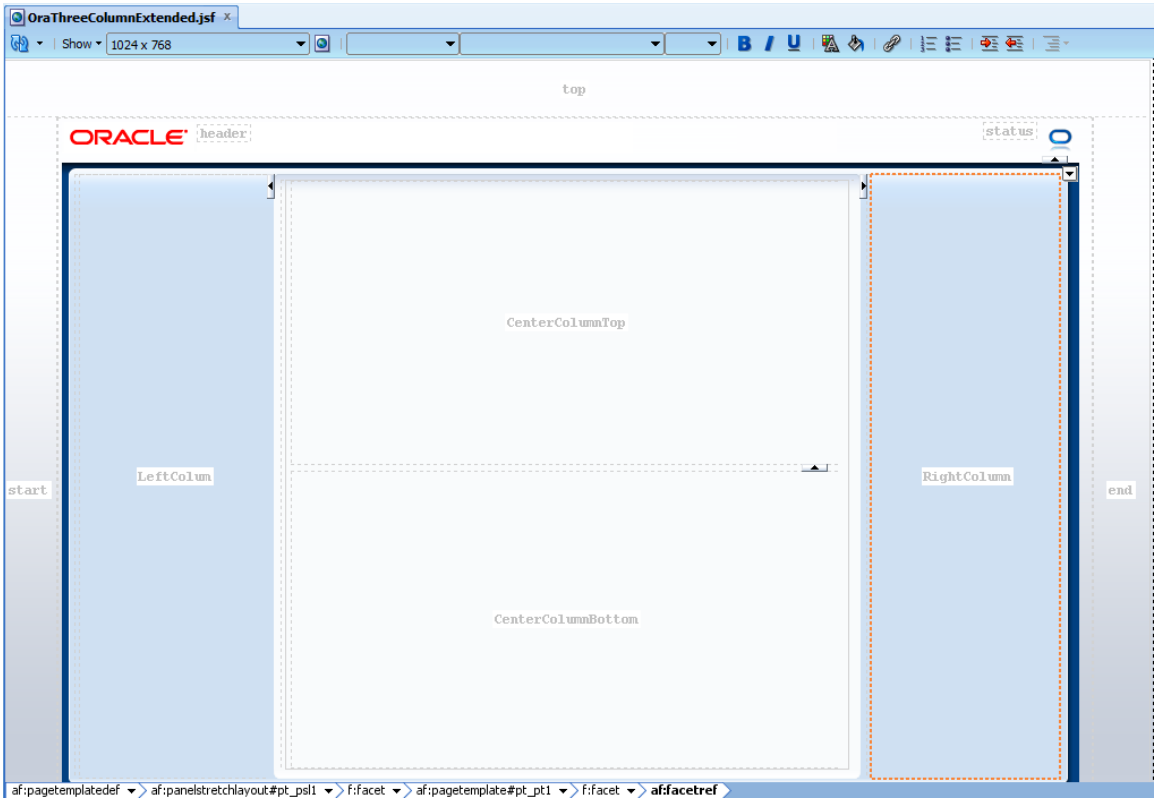


For the application developer who uses this custom page template to add application content, **Facet** references need to be added to the layout.



The three column layout template has a let area, right area and a center area, which we split up into a top center area and bottom center area. For each of these areas, Facets were created earlier.

Note that even if the template you extend already exposes facets for developers to add content, you need to define your own facets and use facet references you add to these areas. The reason for why you have to add custom facet references to existing facets of the extended template is that only facets defined on your template are exposed to a page.

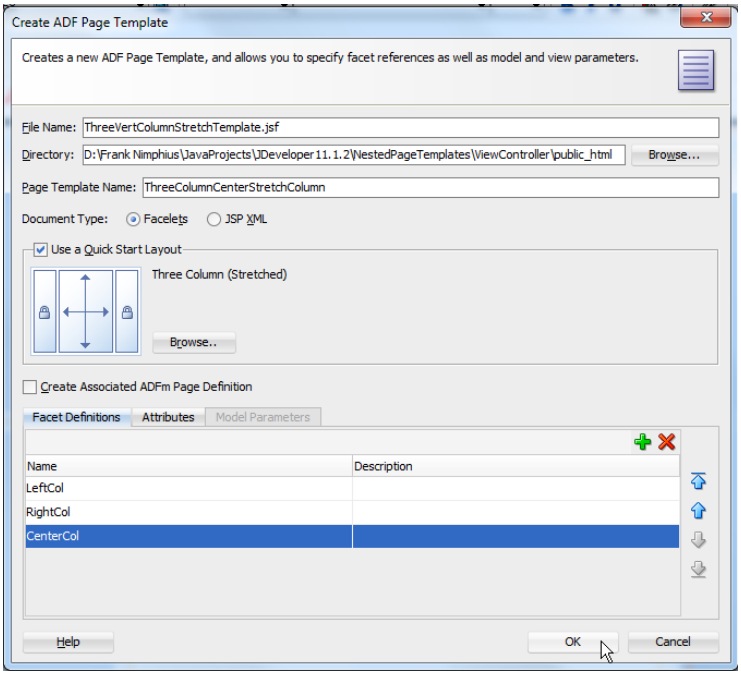


Also note that in previous versions of JDeveloper 11g, the **Facet** component palette entry was called **FacetRef**.

This is all that needed to be done for creating the base template. Save the work and move on to build another template to nest in the base template when building the page.

Creating another template

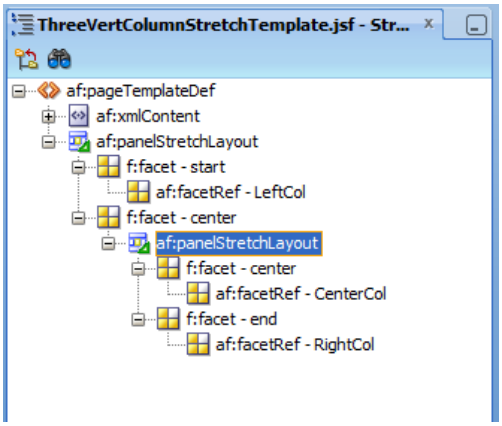
Following the approach explained before, a second template is built that shows three vertical column using the **Quick Start Layout** help.



As before, to define areas for the application developers to add application content to, **Facets** need to be created.

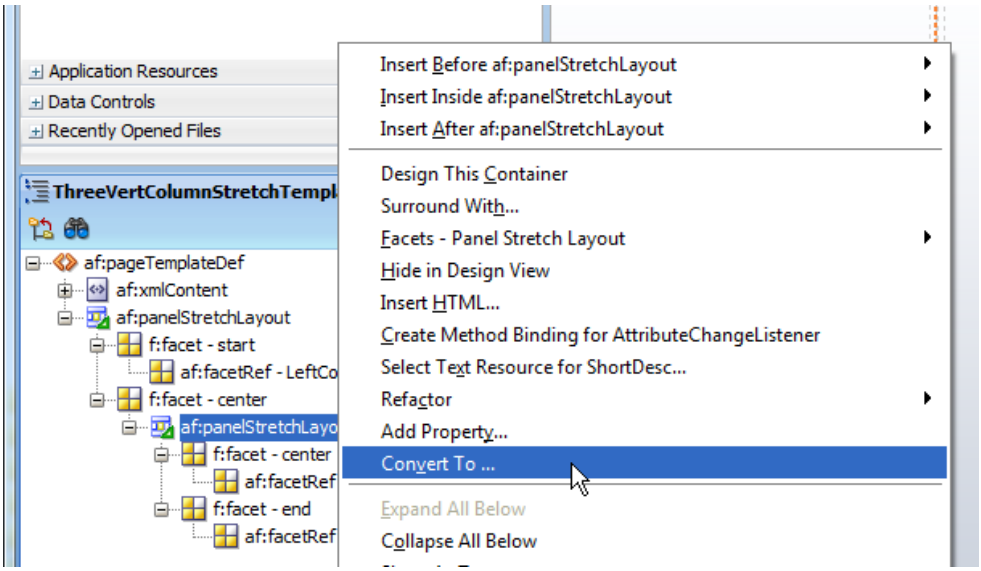
As shown in the image below, the new template definition uses an `af:panelStretchLayout` component, which is a component that ensures the center area and the right column area to stretch. However, using this layout, the right column is not resizable.

For this article, let's pretend that we want the right column to be resizable, in which case the panel stretch layout needs to be replaced with an `af:panelSplitter` component.

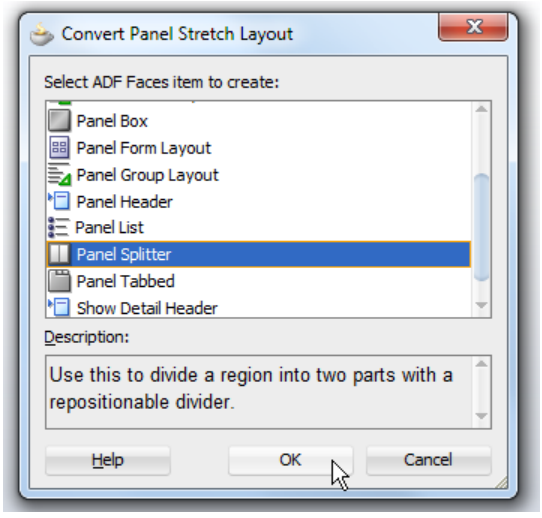


Note that the `af:panelStretchLayout` contains two `af:facetRef` components, which also could have been other ADF Faces UI components. When replacing the panel stretch layout with a panel splitter component, the facetRef components need to be moved to an equivalent location in the panel splitter.

To change the component from `af:panelStretchLayout` to `af:panelSplitter`, select it in the Structure Window and choose **Convert To** from the context menu.

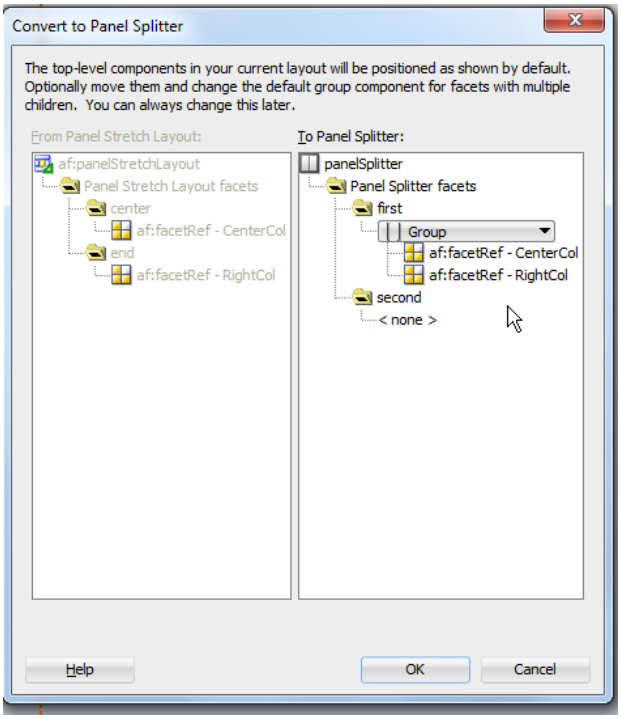


In the opened dialog, choose the target layout component, which in our case is the panel splitter component.

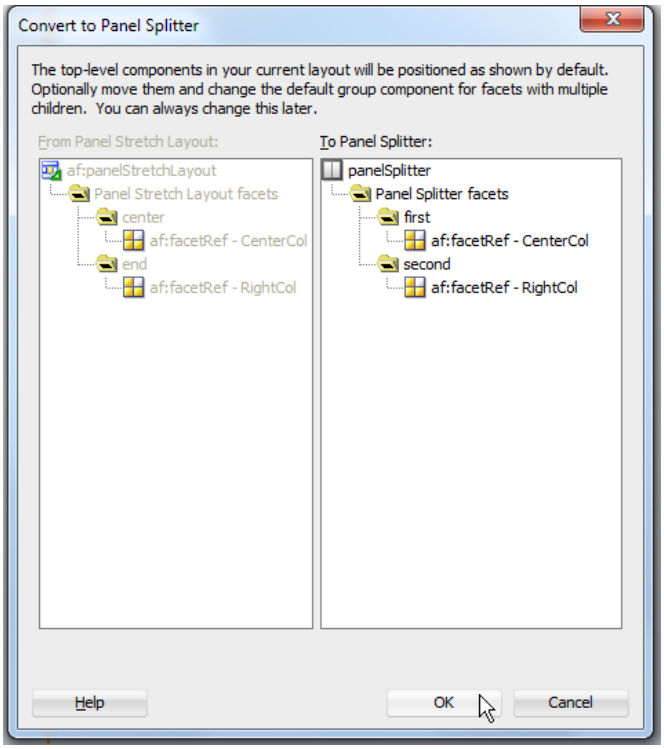


The next dialog also is a new feature in Oracle JDeveloper 11g R2 and allows you to move components contained in the source component to facets of the target component.

In the sample, the two `af:facetRef` components need to be relocated.

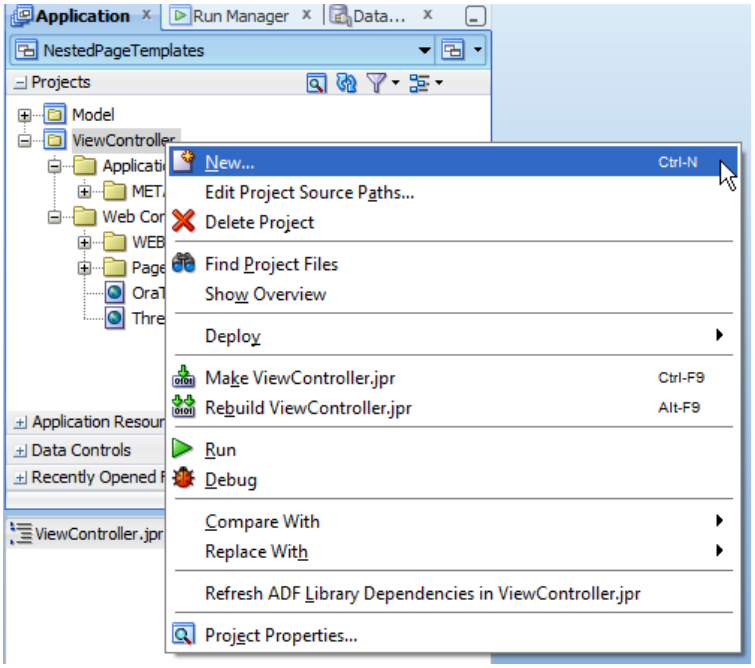


In a first attempt, JDeveloper moved the facets to the panel splitter **first** facet. Using drag and drop, this can be changed as shown below.

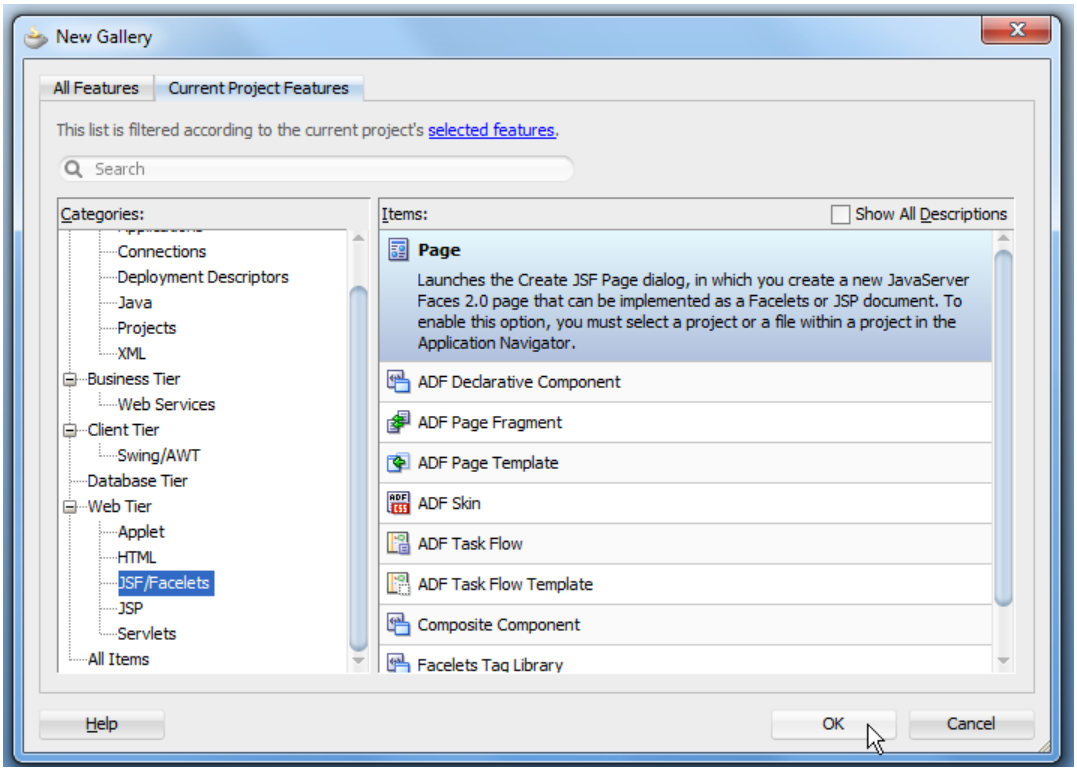


Creating the page

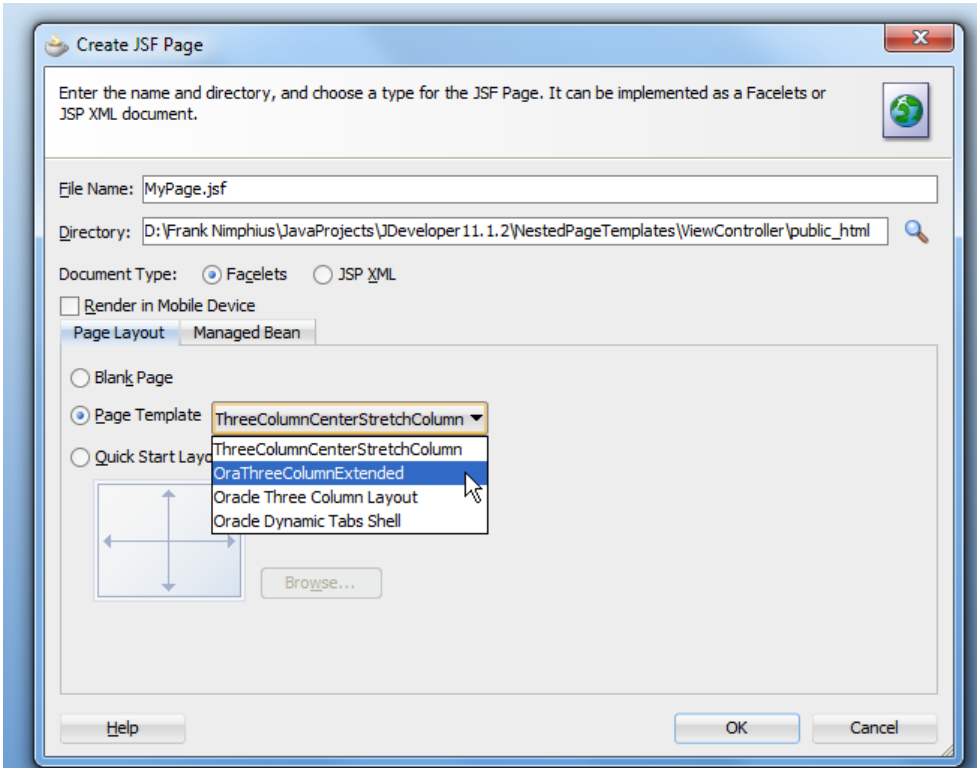
Build a new page by choosing **File | New** from the JDeveloper menu, or using the context menu as shown below.



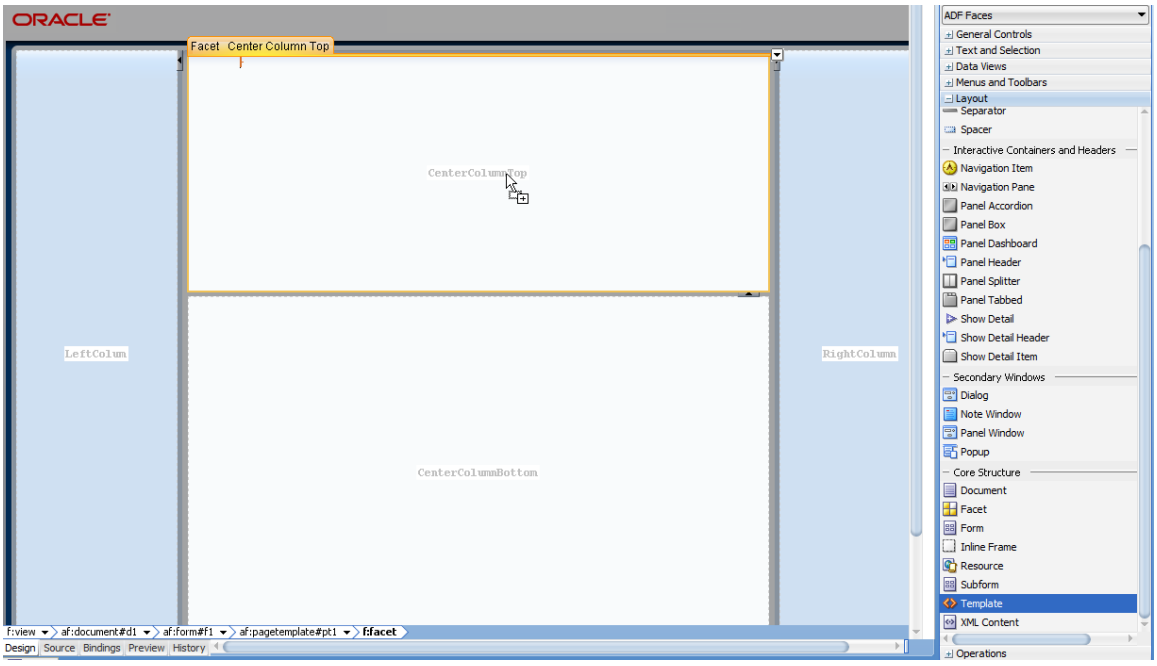
Select ADF Page in the JSF section of the **New Gallery** dialog.



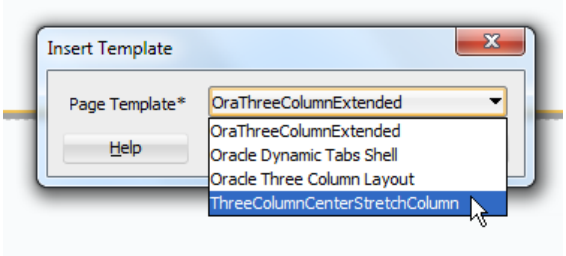
Choose the custom template that extends the **Oracle Three Column Layout** template from the list of available templates to build the page



This template renders the page with three columns, with the center column split up into two parts. In a next step, you are going to further refine the page layout by adding the second template you built as a nested template to the upper area of the center column.



For this, drag the **Template** component entry from the **ADF Faces | Layout** menu to the CenterColumnTop facet.



In the **Insert Template**dialog, select the other template you created when following the steps outlined in this article.

The final page layout now is built out of the **Oracle Three Column Layout** template, a template that extends this template and a nested template.

You can now add components to the **Facet** areas and test the page layout. Because templates are referenced at runtime, and not compiled into a page, changes applied to one of the templates will immediately show in the page.

Note: Make sure when changing any of the templates that the names of the facets don't change and no facetRef is removed that contains content.

Summary and Best Practices

Being able to extend existing templates and nest template in pages – plus the ability to add more than one template definition to a page – is a big win in Oracle JDeveloper 11g R2 that works for JSPX documents and Facelets documents used in this sample.

Be aware that not everything that technically appears possible makes sense in a production environment. The price you pay for deeply nested page templates and complex page layout structures is in performance, because it takes longer for complex pages to load and adjust than for simple structured pages. So before

going too fine granular in defining templates that you then nest infinitively deep, consider performance and ask yourself if the price you pay for complexity is justified by the result you get. If so, go ahead and use templates as you please.

If not, use common sense to judge how fine granular and deep nested your templates should become to achieve a specific layout. For example, I wouldn't recommend using template references for data driven layouts. For such use cases I would have a look at dynamic declarative components or JSF 2.0 composite components.

RELATED DOCOMENTATION

⊠	JDeveloper 11g R2 new features document http://www.oracle.com/technetwork/developer-tools/jdev/jdev-11gr2-nf-404365.html
⊠	Page templates in Web Developer Guide http://download.oracle.com/docs/cd/E16162_01/web.1112/e16181/af_reuse.htm#CACCFCJC
⊠	Page template tag doc http://download.oracle.com/docs/cd/E16162_01/apirefs.1112/e17491/tagdoc/af_pageTemplate.html