ADF Code Corner

Oracle JDeveloper OTN Harvest 07 / 2011

twitter.com/adfcodecorner

Abstract:

The Oracle JDeveloper forum is in the Top 5 of the most active forums on the Oracle Technology Network (OTN). The number of questions and answers published on the forum is steadily increasing with the growing interest in and adoption of the Oracle Application Development Framework (ADF).

The ADF Code Corner "Oracle JDeveloper OTN Harvest" series is a monthly summary of selected topics posted on the OTN Oracle JDeveloper forum. It is an effort to turn knowledge exchange into an interesting read for developers who enjoy harvesting little nuggets of wisdom.

http://blogs.oracle.com/jdevotnharvest/

Author: Frank Nimphius, Oracle Corporation
twitter.com/fnimphiu
30-JUL-2011

Oracle ADF Code Corner OTN Harvest is a monthly blog series that publishes how-to tips and information around Oracle JDeveloper and Oracle ADF.

Disclaimer: ADF Code Corner OTN Harvest is a blogging effort according to the Oracle blogging policies. It is not an official Oracle publication. All samples and code snippets are provided "as is" with no guarantee for future upgrades or error correction. No support can be given through Oracle customer support.

If you have questions, please post them to the Oracle OTN JDeveloper forum:
http://forums.oracle.com/forums/forum.jspa?forumID=83

July 2011 Issue – Table of Contents

ADF Summit Forms to ADF case study available

A new ADF sample application and its associated documentation has been released to the Oracle JDeveloper site on OTN this week:

http://www.oracle.com/technetwork/developer-tools/jdev/index-098948.html

ADF Summit is a case study in redeveloping an Oracle Forms application to Oracle ADF and is based on the Oracle Forms "Summit Sports Good" training application. You get an overview of the project by watching the Camtesia recording Grant Ronald produced and in which he shows how the Oracle Forms application looked and how the redeveloped ADF application looks:

http://download.oracle.com/otn_hosted_doc/jdeveloper/11gdemos/SummitADF/SummitADF.html

In a related whitepaper, Grant Ronald and Lynn Munsinger explain the considerations they followed when building ADF Summit, as well as the design decisions they made:

http://download.oracle.com/otn_hosted_doc/jdeveloper/11gdemos/SummitADF/SummitADF_Redevelopment.pdf

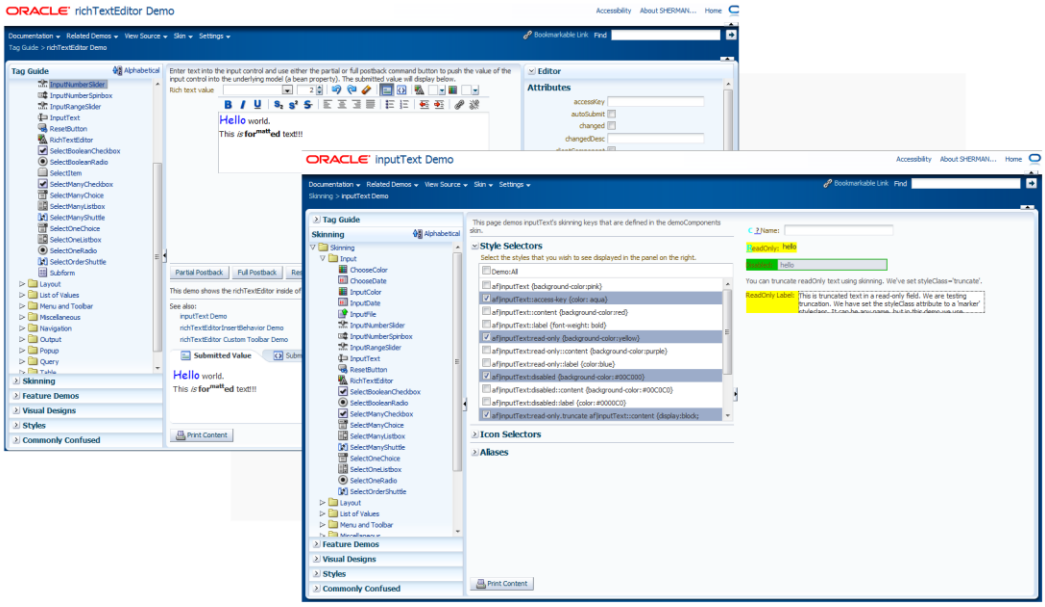
The sample application and install script is available from here:

http://download.oracle.com/otn_hosted_doc/jdeveloper/11gdemos/SummitADF/SummitADFV1_0_08072011.zip

Download of Skin Editor and ADF Faces Component Demo

The download address for the stand alone ADF Faces skin editor and the ADF Faces component demo may not be easy to find. Therefore, we provide the link here in this OTN Harvest summary

http://www.oracle.com/technetwork/developer-tools/adf/downloads/index.html



The ADF Faces component demo is a downloadable WAR file, which you can open in JDeveloper. For this, create a new JDeveloper workspace and then choose **File | New** from the JDeveloper menu. Then, choose the **All Technologies** tab and select **General | Projects | Project from WAR File**. After you imported the WAR file, edit the project properties to suppress compilation when you run the sample. For this, double click onto the project node and select the **Run / Debug / Profile** node. Press the **Edit** button and navigate to the **Tool Settings** node. Unselect the **Make Project** entry and close the dialog pressing OK. Now you can run the index page to play with the ADF Faces components and review the implementation code of the demo.

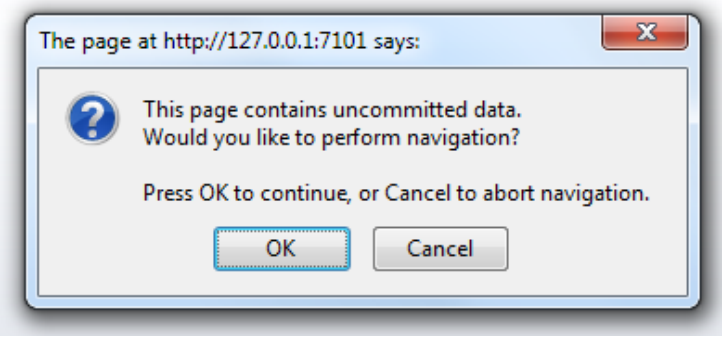
New client behavior tag - af:checkUncommittedDataBehavior

In Oracle JDeveloper 11.1.2, a new client behavior tag `af:checkUncommittedDataBehavior` is provided to check for uncommitted data when navigating away from a page using a command button that has its **immediate** property set to true. The tag can be applied as a child of any command component, like

- `af:commandButton`
- `af:commandLink`
- `af:commandMenuItem`
- `af:commandToolbarButton`
- …

http://download.oracle.com/docs/cd/E16162_01/apirefs.1112/e17491/tagdoc/af_checkUncommittedDataBehavior.html

For the client behavior to work, you must set the `document` tag's **uncommittedDataWarning** attribute to **on**.



Note that the same tag also is available in **Oracle JDeveloper 11.1.1.5**. Though the ADF Faces tag documentation lacks this information, the tag itself is shown in the ADF Faces Component Palette (ctrl+shit+P) within the **ADF Faces** accordion.

favicon and browser bookmark icons in JDeveloper 11.1.2

The **favicon** is the little icon that displays in the Browser URL address field when a requested page loads. In Oracle JDeveloper 11.1.1.x releases, the **favicon** needed to be added to the page source. In JDeveloper

11.1.2 a new attribute, **smallIconSource** has been added to the `af:document` tag to serve the favicon easily.

See: http://download.oracle.com/docs/cd/E16162_01/apirefs.1112/e17491/tagdoc/af_document.html

smallIconSource

Specifies a small icon that the browser may insert into the address bar (commonly known as a "favicon"). If this attribute is not specified, the browser may default to using a file named "favicon.ico" located at the root of your server. (This default behavior is not something provided by this framework and may vary between browsers.) This attribute supports a space-delimited list of files (each file may be wrapped in quotes or apostrophes if the file path contains a space). If the file path specifies a single leading slash, this means that the file is located inside of the web application's root folder (so "/small-icon.png" would resolve to something like "http://www.oracle.com/adf-faces/small-icon.png"). If the file path specifies 2 leading slashes, this means that the file is located inside of the server's root folder (so "//small-icon.png" would resolve to something like "http://www.oracle.com/small-icon.png"). Browsers typically expect these files to be 16 pixels by 16 pixels. Typically, the first listed file will be the one used. Otherwise, if a browser only supports certain kinds of files, the first file in the list that uses a supported file extension will be the one that is used for that browser.

Another new attribute is the **largeIconSource** that applies an image to bookmark entries in borwsers.

largeIconSource

Specifies a large icon that the browser may use when bookmarking this page to your device's home screen. If this attribute is not specified, the browser may default to using a file named "apple-touch-icon.png" located at the root of your server. (This default behavior is not something provided by this framework and may vary between browsers.) This attribute supports a space-delimited list of files (each file may be wrapped in quotes or apostrophes if the file path contains a space). If the file path specifies a single leading slash, this means that the file is located inside of the web application's root folder (so "/large-icon.png" would resolve to something like "http://www.oracle.com/adf-faces/large-icon.png"). If the file path specifies 2 leading slashes, this means that the file is located inside of the server's root folder (so "//large-icon.png" would resolve to something like "http://www.oracle.com/large-icon.png"). Browsers typically expect these files to be 57 pixels by 57 pixels but could be larger, e.g. 72 pixels by 72 pixels or 129 pixels by 129 pixels. Typically, the first listed file will be the one used. Otherwise, if a browser only supports certain kinds of files, the first file in the list that uses a supported file extension will be the one that is used for that browser.

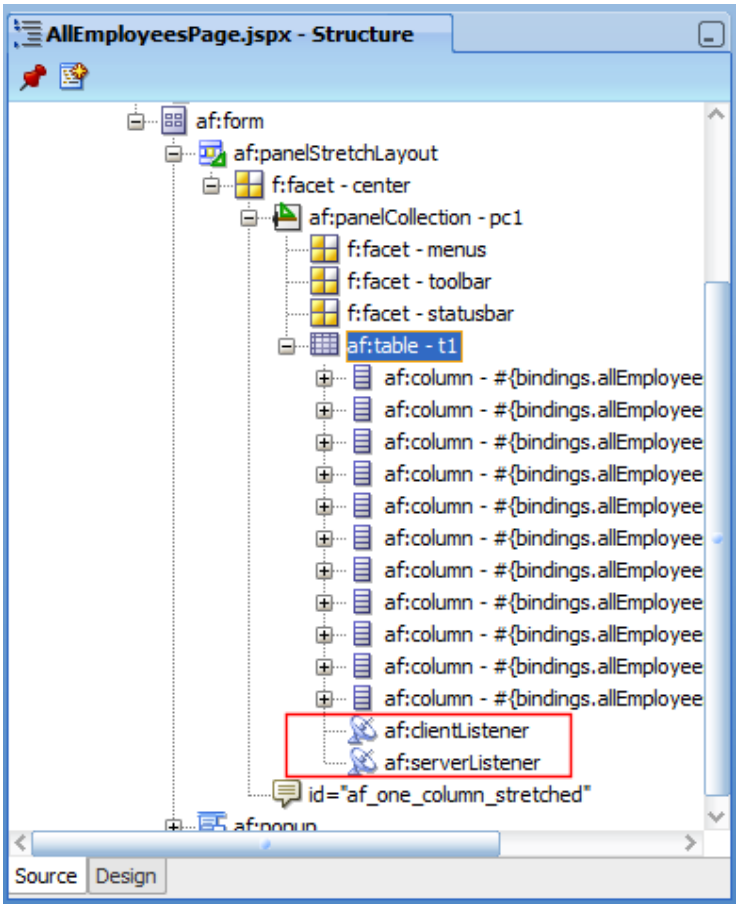
```
<af:document title="My Page"
             smallIconSource="/favicon.png /favicon.ico"
             largeIconSource="/touchicon.png">
    <af:form> ... </af:form>
</af:document>
```

Access component that queued a custom client event

In ADF Faces, to invoke a server side method in a managed bean, you use the `af:serverListener` tag. The `af:serverListener` tag is added as a child to the component that owns the event and called from JavaScript in a call to `AdfCustomEvent.queue( … )`

In this example, the af:serverListener is added to a table to notify a manage bean method about a double-click action.

```
<af:table ...>
  <af:column> ... </af:column>
   ...
  <af:clientListener method="handleTableDoubleClick"
                     type="dblClick"/>
  <af:serverListener type="TableDoubleClickEvent"
                      method="#{myBean.handleTableDoubleClick}"/>
</af:table>
```



The JavaScript function that is called by the `af:clientListener` is shown next.

```
function handleTableDoubleClick(evt){
  var table = evt.getSource();
  AdfCustomEvent.queue(table, "TableDoubleClickEvent",{}, true);
  evt.cancel();
}
```

The first argument in the call to `AdfCustomEvent.queue` represents the event owner, the table component. This information is passed to the managed bean method, which has the following signature.

```
public void handleTableDoubleClick(ClientEvent ce){
        RichTable richTable = (RichTable) ce.getComponent();
```

```
        //... work with rich table component
}
```

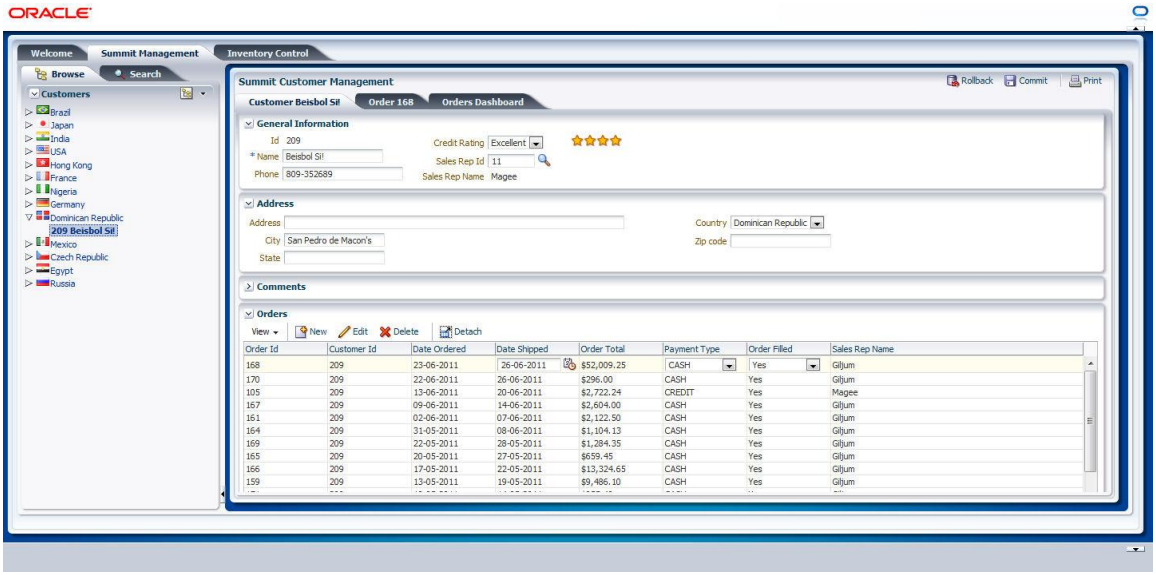
As you can see, there is no need to look up the event owning component by searching the JSF `UIViewRoot` with or without help of JSFUtils.

How-to open a page template served from an ADF library

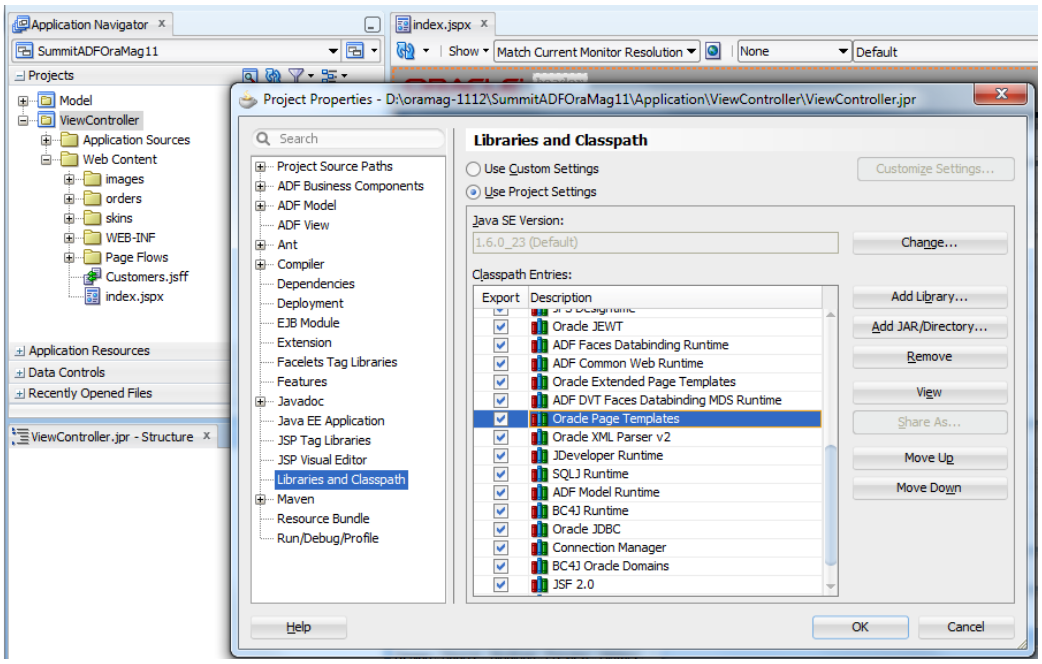
Page templates in Oracle ADF Faces can be defined within an application's View Controller project or deployed from and referenced in an ADF library.

Best practices for reuse in Oracle ADF is to deploy the page template in an ADF library, in which case the template sources are not visible in the Application Navigator. But if the template source is not located in the project, how can you access the template source file to have a look, e.g. to better understand how it is constructed or which components to skin to change the look and feel.

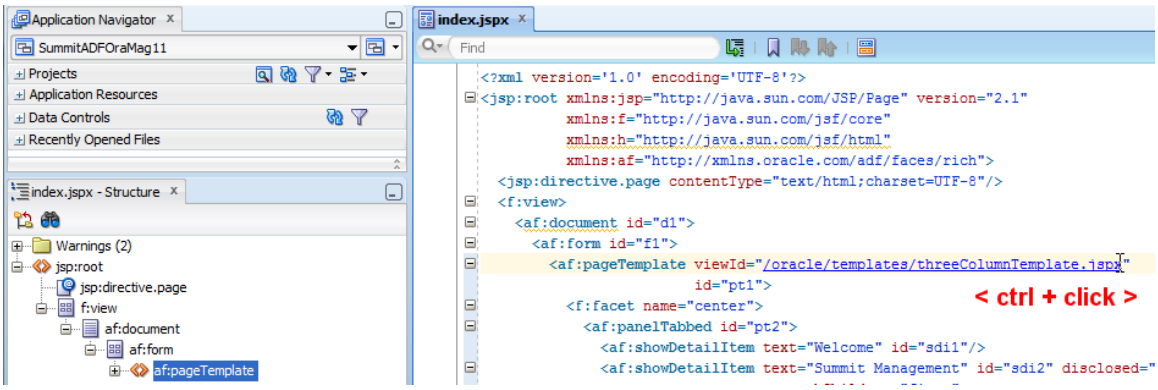
For example, the ADF Summit application shown below uses the "three column template" that comes with Oracle JDeveloper.



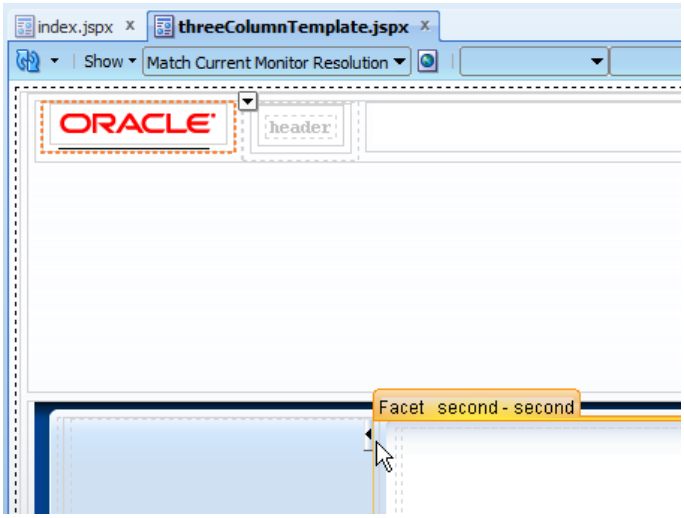
The three column template is contained in the **Oracle Page Templates** library, which gets added to the **ViewController** project when one of its contained templates is selected for a page.



To open the three column template sources in Oracle JDeveloper, open the page that references the template in the Oracle JDeveloper code view as shown in the image below. Then press the **ctrl** key and **click** onto the **pageTemplate viewId** value.



Using the ADF Faces resource loader, Oracle JDeveloper loads the template definition from the ADF library to show it in read-only mode.



Using JavaScript in ADF Faces

ADF Faces exposes a client side JavaScript API that allows you to work on the client side as you would on the server side using JavaServer Faces component object. Using the browser side Document Object Model (DOM) to access UI object rendered on a screen is not a good choice to use as it does not know about ADF Faces components and their behavior but about HTML markup. Working with the JavaScript APIs exposed on the ADF Faces client architecture ensures your code works across browser types and versions and that it well integrates with the JavaServer Faces request lifecycle. The same APIs are used internally by the ADF Faces component developers, for example to code client side validation and input conversion, or to implement component behavior. Before you start any JavaScript work in ADF Faces, take the time to read the whitepaper referenced below to gain a better understanding of how to work with JavaScript in ADF Faces and what is best practices

http://www.oracle.com/technetwork/developer-tools/jdev/1-2011-javascript-302460.pdf

The ADF Faces JavaScript APIs are documented as part of the Oracle Fusion Middleware API documentation

http://download.oracle.com/docs/cd/E21764_01/apirefs.1111/e12046/toc.htm

How-to access the column value of the selected table row

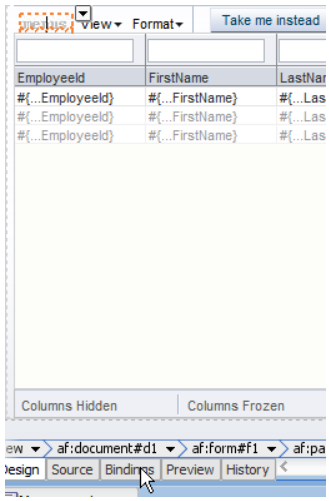
A common use case is to access an attribute value of a selected row in a table configured for single row selection using Java or Expression language. Usually the requirement is that the selected row column data serves as the input value for another action, for example the input parameter of a bounded task flow.

The table in ADF Faces is stamped when rendered, which means that rows in a table are not objects that can be accessed directly. There are ways to programmatically access the selected row in a table, which is good to know when working with multi row select tables, but using an attribute value binding in the ADF binding layer is a lot easier than this and also EL accessible.

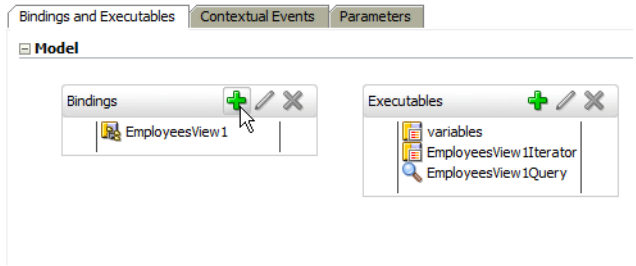
The table, when created by dragging a collection from the Data Controls panel, has the SelectionListener property of the table configured to point to the ADF tree binding that populates the table.

When a user clicks into a table row, then this SelectionListener ensures the iterator in the ADF binding layer is synchronized so that the selected table row becomes the current row in the iterator. An iterator in ADF however is a collection of objects, for which the current row is like a window to the selected data. This window can be extended with an additional view so to say, which is an attribute value binding.

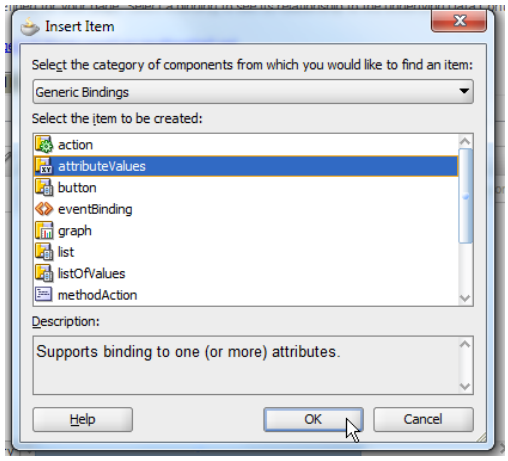
On the JSF page that contains the table, click onto the **Bindings** tab to switch to the binding editor that updates the page's PageDef file.



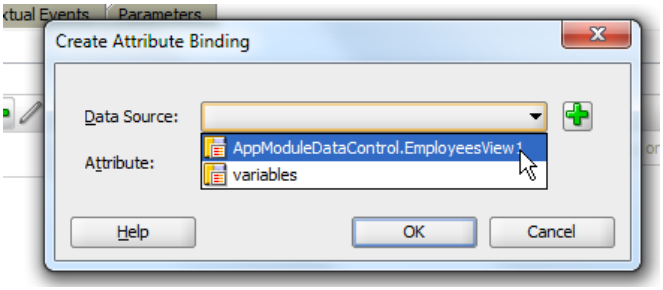
In the bindings dialog, click the green plus icon to launch the dialog to manually create additional bindings. Note the EmployeesView1Iterator that is shown in the **Executables** section.



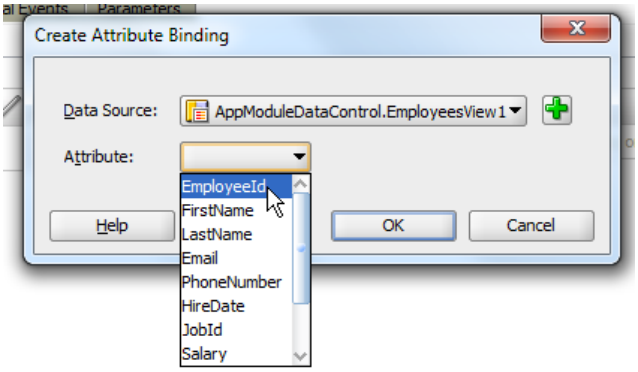
In the Insert Item dialog, choose **attributeValues** to create a new attribute binding, which then represents the column value for a selected row – a cell value.



In the **Data Source** property, select the iterator that populates the table, EmployeesView1Iterator in this sample



In the **Attributes** list, select the attribute which value you need to pass on or access. **Ok** the dialog to create the binding.



For the sample above, in which the attribute value binding is created for the EmployeeId attribute, the value of the selected table row for this column is EL accessible with

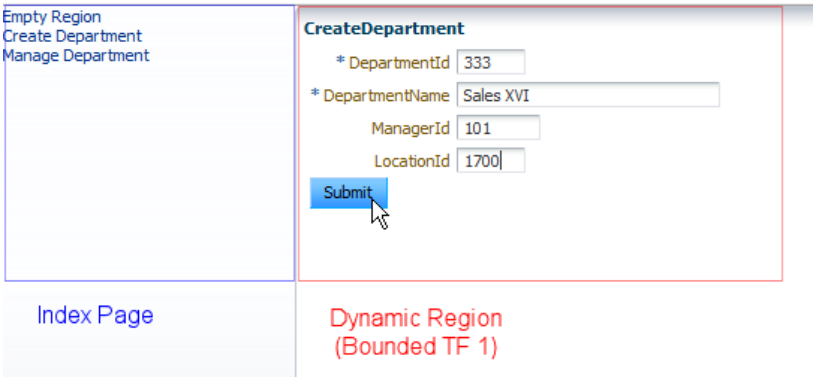
#{bindings.EmployeeId.inputValue}

If you needed to access this information from Java, then you do as follows

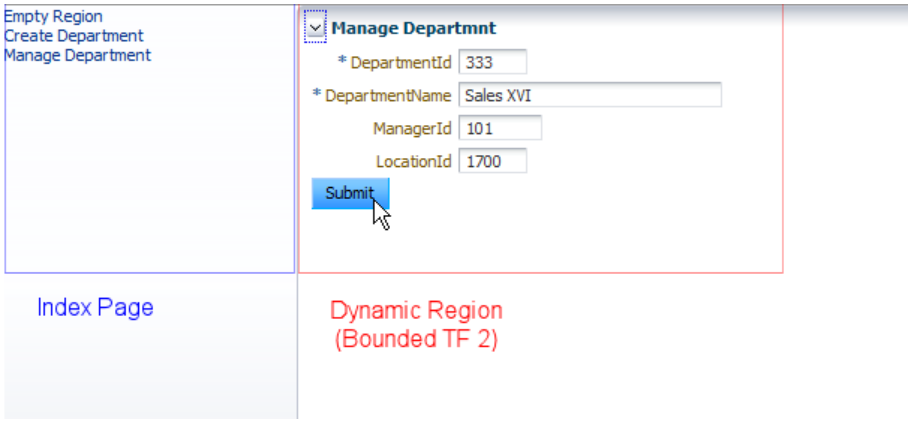
```
BindingContext bctx = BindingContext.getCurrent();
BindingContainer bindings = bctx.getCurrentBindingEntry();
AttributeBinding employeeId  =
                (AttributeBinding) bindings.get("EmployeeId");
Oracle.jbo.domain.Number idValue =
                (Oracle.jbo.domain.Number) emmployeeId.getInputValue();
```

How to switch content of dynamic region from within region

The use case is as shown in the images below. A JSPX index page has a menu bar with command links to switch the content of a dynamic region shown on the right to the menu. Clicking a command link either shows an empty task flow, the Create Department task flow or the Manage Department task flow.



When the **Create Department** link is clicked, the dynamic region switches to the bounded task flow to create a new department. The requirement now is that when the new department is created by pressing or clicking the **Submit** button in the page fragment displayed in the region, the dynamic ADF region should show the task flow that manages departments instead.



The same could be achieved by the user pressing the **Manage Department** command link. In this case however it should be done programmatically in the context of the new department creation.

A generic solution for this is use case is to use contextual events, in which a message is broadcasted through the binding layer for the index page to invoke the **Manage Department** action

http://download.oracle.com/otn_hosted_doc/jdeveloper/11gdemos/AdfInsiderContextualEvents/AdfInsiderContextualEvents.html (see a Video)

http://www.oracle.com/technetwork/issue-archive/2011/11-may/o31adf-352561.html (read Article)

However, we want to look for another solution to achieve the same:

The command links shown in the image above are configured reference a managed bean to invoke a method that switches the content of the AD region.

```
<af:panelGroupLayout id="pgl1" layout="vertical">
   <af:commandLink text="Empty Region"
                   action="#{viewScope.DynRegionBean.empty}"
                   id="cl3" partialSubmit="true"/>
   <af:commandLink text="Create Department"
                    action="#{viewScope.DynRegionBean.createDepartmentsTF}"
```

```
                    id="cl1" partialSubmit="true"/>
  <af:commandLink text="Manage Department"
                    action="#{viewScope.DynRegionBean.manageDepartmentTaskFlow}"
                    id="cl2"/>
</af:panelGroupLayout>
```

To learn about and how-to create dynamic regions, read up in the product documentation here:

http://download.oracle.com/docs/cd/E21764_01/web.1111/b31974/taskflows_regions.htm#CHDJHACA

Note that the scope of the manage bean that is created for switching the task flow displayed in the ADF region is changed from backingBeanScope (default) to viewScope. When working with ADF bound pages in a bounded task flow you should do the same. The default – unfortunately - is only of limited use.

The managed bean that is referenced from the ADF task flow binding in the ADF PageDef file of the Index page also needs to be updated accordingly

```
<executables>
   <variableIterator id="variables"/>
   <taskFlow id="dynamicRegion1"
         taskFlowId="${viewScope.DynRegionBean.dynamicTaskFlowId}"
         activation="deferred"
         xmlns="http://xmlns.oracle.com/adf/controller/binding"/>
 </executables>
```

You don't have to perform the change in the source editor but can use the JDeveloper Property Inspector for this, which may be a more safe way of doing it. For this, select the PageDef file in the Application Navigator and select the task flow binding in the Structure Window. Then open the Property Inspector.

The managed bean also is slightly changed from the default as we wanted it to support an empty region, which is what is shown before users click on **Create Department** or **Manage Department**. The code is as shown below

```
public class DynRegionBean {
    //initially the ADF region shows empty
    private String taskFlowId ="";

    public DynRegionBean() {}

    //method queried from the task flow binding to set the bounded
    //task flow reference
    public String getDynamicTaskFlowId() {
      return taskFlowId;
    }


    //method called from the Create Department link
    public String createDepartmentsTF() {
     taskFlowId =
         "/WEB-INF/CreateDepartmentsTF.xml#CreateDepartmentsTF";
```

```
        return null;
    }


    //method called from the Manage Department link to show the task
    //flow that manages the current department
    public String manageDepartmentTaskFlow() {
     taskFlowId =
      "/WEB-INF/ManageDepartmentTaskFlow.xml#ManageDepartmentTaskFlow";
      return null;
    }


  //method calle by the "Empty" command item to switch the ADF region
  //back to show no task flow
  public String doEmpty(){
      taskFlowId ="";
      return null;
    }
}
```

Whenever one of the command links is pressed, the ADF region is refreshed using PPR configured on its PartialTrigger property.

```
<af:region value="#{bindings.dynamicRegion1.regionModel}" id="r1"
           partialTriggers="::cl1 ::cl2 :: cl3"/>
```

When refreshing, the ADF region queries the task flow binding for the task flow to display in the region, which then always shows the task flow determined by the link pressed.

An alternative **solution to the use case** introduced earlier is to programmatically queue the **Manage Department** command link action from the **Submit** button in the **Create Department** region.

Caution: If the Create Department bounded task flow is supposed to be reused in other applications, then the **Manage Department** command link's id needs to be passed as an input parameter to the task flow shown as the **Create Department** region. Or, you turn away from this recipe and use contextual events instead.

When the **Submit** button in the **Create Department** region is pressed, the following Java code is exeuted to simulate the action invoked when a user pressed the **Manage Department** link.

```
public void onDepartmentCreate(ActionEvent actionEvent) {
  //find RichRegion
   UIViewRoot root = FacesContext.getCurrentInstance().getViewRoot();
   RichCommandLink commandItem =
                  (RichCommandLink)root.findComponent("cl2");
   if(commandItem!= null){
       ActionEvent event = new ActionEvent(commandItem);
        event.queue();
```

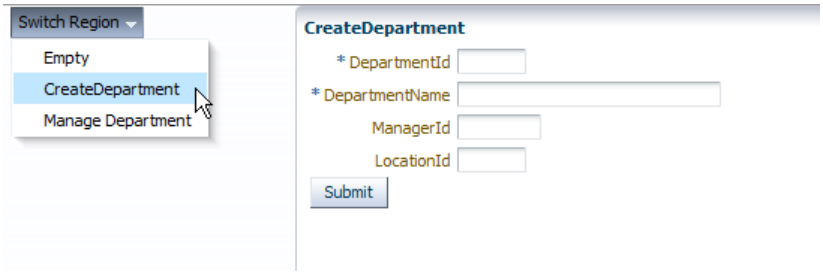
```
    }
}
```

As mentioned, if you want to reuse the Create Department bounded task flow, then the command link reference "cl2" should be passed as an input parameter. Also be aware of naming containers in JavaServer Faces. If, for example, the menu components are surrounded by an **af:subForm** component to avoid form validation when a menu item is pressed, without setting immediate="true" on the command item, then this wraps the command links in a naming container. If e.g. the subForm id was "s1" then the command link Id changes to s1:cl1. If you are unsure of whether or not a container in a page is a naming container, have a look in the ADF Faces tag documentation:

http://download.oracle.com/docs/cd/E21764_01/apirefs.1111/e12419/toc.htm

The code shown above looks the command link up in the Faces view root and, if found, queues the command component's action for processing.

Note: Another option to build a menu structure for switching with a dynamic region is the `af:menuBar` and the `af:commandMenuItem` components as shown below:



The only difference here is in the code used to queue the action, which now doesn't work with a `RichCommandLink` instance but a `RichCommandMenuItem` instance to queue the action. Again, pressing the command button queues the event on the command item in the index page as if a user clicked it.

RELATED DOCOMENTATION

⊠	ADF Insider http://www.oracle.com/technetwork/developer-tools/adf/learnmore/adfinsider-093342.html
⊠	ADF Insider Essentials http://www.oracle.com/technetwork/developer-tools/adf/learnmore/adfinsideressentials-337133.html
⊠	ADF Code Corner http://www.oracle.com/technetwork/developer-tools/adf/learnmore/index-101235.html
⊠	OTN Harvest http://blogs.oracle.com/jdevotnharvest/

☒	Steve Muench's not yet documented samples http://blogs.oracle.com/smuenchadf/resource/examples
☒	"Decompiling ADF Binaries" blog (ADF BC centric) http://jobinesh.blogspot.com/
☒	Fusion Developer Guide Product Documentation http://download.oracle.com/docs/cd/E21764_01/web.1111/b31974/toc.htm
☒	Fusion Web User Interface Developer Guide Product Documentation http://download.oracle.com/docs/cd/E21764_01/web.1111/b31973/toc.htm
☒	Skin Editor Developer Guide http://download.oracle.com/docs/cd/E16162_01/user.1112/e17456/toc.htm