ADF Code Corner

002. ADF Programmer Cheat Sheet 2010

twitter.com/adfcodecorner

Abstract:

Some solutions are too small to blog or write an article about. I created this page for code examples like these that don't require a sample workspace. This way nothing gets lost!.

Author: Frank Nimphius, Oracle Corporation
twitter.com/fnimphiu
01-JAN-2010 – 31-DEC-2010

ADF Faces Solutions by Component

af:declarativeComponent

Problem: A command component in a declarative component should return an outcome for navigation. For this I use the action property with a static outcome. This works when added to a task flow. Now I want to make the outcome dynamic in that an attribute on the declarative component is used to provide the outcome string. How to go about this ?

Solution: The action property of a command component expects a String or, if an EL expression is used, a method expression. The attribute reference in a declarative component is not a method expression and therefore a solution would be to use a managed bean to access the attribute to read the outcome string in.

The following code is a managed bean method that you could referencee from the command component's action property. Configuring the declarative component project to include the Page Flow libraries, you can set the managed bean scope to "backingBean" so that multiple declarative components can be added to a single page without conflicts.

```
public String commandAction(){

 String outcome = null;
 //lookup the component instance and the attribute
 FacesContext fctx = FacesContext.getCurrentInstance();
 ELContext elctx = fctx.getELContext();
 Application app = fctx.getApplication();
 ExpressionFactory efactory = app.getExpressionFactory();

 ValueExpression componentExpr =
 efactory.createValueExpression(
   elctx,"#{component}", Object.class);
 RichDeclarativeComponent _this =
    (RichDeclarativeComponent) componentExpr.getValue(elctx);
 outcome =
     (String) _this.getAttributes().get("theOutcomeAttr");
 return outcome;
}
```

The managed bean method - the action method - uses EL to lookup the declarative component instance, which it does so by the "component" variable, which is a default setting created when you build a new declarative component (have a look at the declarative component's var property). From here you have access to all the attributes (the

standard component attributes and the custom attributes) to read in. The string you get from the custom attribute you define for passing in the outcome string is what you return from the method. Because custom attributes of a declarative component can use EL itself, for example to reference other managed beans, you are dynamic in what the declarative component navigation outcome is.

af:popup

Problem: **How to open a popup dialog in ADF Faces RC?**

Solution: In releases before Oracle JDeveloper 11.1.1.3 (aka. JDeveloper 11g R1 PS2) developers needed to use the af:showPopupBehavior operation tag or JavaScript to open a DHTML dialog for a view. With the new release, a Java API is available that you can use as shown below

```
// assuming the popup should be launched from the action
listener of a button or command link
public void someAcion(ActionEvent event)
{
 ...
 FacesContext fctx = FacesContext.getCurrentInstance();
 UIComponent launcher = (UIComponent) event.getSource();
 //get the client id of the launching component to
 //add as a hint argument to the popup
 String alignId launcher.getClientId(fctx);
 //Assuming that the popup has a JSF component binding
 //to the managed bean. Use the "binding" attribute of
 //the af:popup to build this binding reference
 RichPopup popup = getPopup();
 RichPopup.PopupHints hints = new RichPopup.PopupHints();
 //align the popup with the component launching it
 hints.add(
    RichPopup.PopupHints.HintTypes.HINT_ALIGN_ID, launcher)
 hints.add(
    RichPopup.PopupHints.HintTypes.HINT_LAUNCH_ID, launcher)
 hints.add(
      RichPopup.PopupHints.HintTypes.HINT_ALIGN,
      RichPopup.PopupHints.AlignTypes.ALIGN_AFTER_END);
 popup.show(hints);
}
```

Problem How to show a confirmation dialog before navigating to another page in an unbounded task flow ?

Solution Assuming a popup dialog is opened - an af:popup containing an af:dialog component - with an Ok and Cancel button displayed (use the Property Inspector on the dialog component to determine the button options). The user can confirm and clos the dialog by clicking one of the two buttons displayed. This selection will raise a DialogEvent in a managed bean if the dialogListener property of the af:dialog component is configured (see: af:dialog tag

documentation)

```
public void confirmAction(DialogEvent dialogEvent) {
  DialogEvent.Outcome result = dialogEvent.getOutcome();
  if (result == DialogEvent.Outcome.ok) {
   NavigationHandler navigationHandler =
              FacesContext.getCurrentInstance()
             .getApplication().getNavigationHandler();
   navigationHandler.handleNavigation(
         FacesContext.getCurrentInstance(),
         null, "controlcase_to_follow");
   }
}
```

When the user closes the dialog with pressing the ok button, then the JavaServer Faces NavigationHandler is called to follow a defined control case. In the example above the control case is called "control_case_to_follow".

Control cases in ADF Faces Controller are the lines you draw between activities, or to an activity from a wild card element. If you use bounded task flows, then you cannot use the NavigationHandler and instead you need to programmatically set the new viewId on the ViewPort that you access through the ControllerContext.

---

af:table

Problem: **How to enforce uppercase character entries in a table filter?**

Solution: There are two possible solutions to this requirement. The first solution is to define a table query listener that uses Java in a managed bean to access the search field values and turn them into uppercase. Though this solution is easy to implement, it does not give immediate feedback to the user. Therefore, a second solution is to use JavaScript for instant to uppercase conversion on the table filter while the user enters data. To implement the JavaScript based solution, you

- Add an af:inputText component to the af:column filter facet
- Set the inputText field value property to #{vs.filterCriteria.<attribute_name>}.
- **Tip:** Use the EL builder in Oracle JDeveloper for this. The "vs" entry is under the "JSP Object" node
- Create a JavaScript method like shown below. Use the af:resource component tag
- Add an af:clientListener to the af:inputTextComponent (see Operations category). In the client listener "METHOD", reference the JavaScript method (just the name, no

arguments or brackets). As the "TYPE", add keyDown.

An example page source is shown below. The table filter is added as an **inputText** field with the **clientListener** added. Note the reference to the JavaScript method "toUpperCase" and the type, which is set to "keyDown"

```
<af:column sortProperty="DepartmentName" filterable="true"
           sortable="false"
           headerText="…" id="c4">
   <af:outputText value="#{row.DepartmentName}" id="ot4"/>
 <f:facet name="filter">
   <af:inputText label="Label 1" id="it1"
                 value="#{vs.filterCriteria.DepartmentName}">
      <af:clientListener method="toUpperCase" type="keyDown"/>
   </af:inputText>
  </f:facet>
</af:column>
```

The JavaScript source added to the page (or referenced from an external JavaScript library) looks as follows

```
<af:resource type="javascript">
   function toUpperCase(evt){
      var _filterField = evt.getCurrentTarget();
      var _value = _filterField.getSubmittedValue();
      _filterField.setSubmittedValue(_value.toUpperCase());
   }
</af:resource>
```

When the user adds a value in the search field the JavaScript function is invoked and turns the filter input to all upper case.

Problem **How to add a new row at the end of the table and scroll the table to display the new row?**

Solution In Oracle JDeveloper 11g, you can create a new row in the iterator that is used to populate the table. Doing so, you get information about the last row in the table to add insert the new row at its end.

```
public String newRowAction() {

   CollectionModel tableModel =
     (CollectionModel) table1.getValue();
   JUCtrlHierBinding adfModel =
           (JUCtrlHierBinding) tableModel.getWrappedData();
   DCIteratorBinding dciter = adfModel.getDCIteratorBinding();

   //get the last row for the index and create a new row for the
   //user to edit
   Row lastRow = dciter.getNavigatableRowIterator().last();
   Row newRow = dciter.getNavigatableRowIterator().createRow();
   newRow.setNewRowState(Row.STATUS_INITIALIZED);

   int lastRowIndex =
    dciter.getNavigatableRowIterator().getRangeIndexOf(lastRow);
```

```
    dciter.getNavigatableRowIterator().insertRowAtRangeIndex(
        lastRowIndex+1, newRow);
    // make the new row the current row of the table
    dciter.setCurrentRowWithKey(newRow.getKey()
             .toStringFormat(true));

    //table should have its displayRow attribute set to
    //"selected"
   AdfFacesContext.getCurrentInstance().addPartialTarget(table1);

  return null;
  }
```

The method above is contained in a managed bean and referenced from the action property of a command button (having its partialSubmit property set to "true"). The table to create the new row in is referenced in the managed bean using the table's binding property. In the example above, the table handle in the managed bean is "table1" (setTable1, getTable1). Note that to scroll to the newly created row, you need to set the display row attribute of the table to show the selected row. By default the table always displays the first row when refreshed.

af:CommandButton

Problem: Invoke a command button from JavaScript on the browser client

Solution: The client side framework in ADF Faces RC allows you to qeue the action of a component to be exected on the server.

```
function callRefreshButton(){
 var buttton1 =
     AdfPage.PAGE.findComponentByAbsoluteId("refreshButton");
 ActionEvent.queue(button1,true);
}
```

When calling this JavaScript function from anywhere on the page, it searches the button by absolute id and queues the action on it. If the button is hidden, then you better use the serverListener component instead.

ADF Binding Solution

Problem: You want to access the ADF binding layer in a managed bean

Solution: Though there are many ways to achieve this goal, the easiest is to use the *BindingContext*

```
import oracle.adf.model.BindingContext;
```

```
import oracle.binding.BindingContainer;
...

BindingContext bindingContext = BindingContext.getCurrent();
BindingContainer bindings =
bindingContext.getCurrentBindingsEntry();
```

The BindingContainer can also be casted to an instance of DCBindingContainer if you prefer to work with typed binding methods

Problem: You want to sequentially execute two binding methods or operations when pressing a command button on an ADF Faces page.

Solution: There is no declarative option to do this. Instead you need to write a managed bean method that expects an ActionEvent as the argument. To create the method, select the command button and choose the ActionListener property. Then click the arrow icon to the right and choose Edit from the menu. The popup dialog allows you to ceate a new managed bean or choose an existing one and then to create the action method. If your command button or link is bound to an ADF operation or method (like Next, Previous etc.) a checkbox is shown that allows you to generate Java calls to replace ths functionality (this way the current functionality is preserved)

In ADF, methods and operations are performed by the OperationBinding object. So to call operations or methods sequentially, you only need to define two operations and call them one after the other

```
import oracle.adf.model.BindingContext;
import oracle.binding.BindingContainer;
import oracle.binding.OperationBinding;
...

BindingContext bindingContext = BindingContext.getCurrent();
BindingContainer bindings =
bindingContext.getCurrentBindingsEntry();
//assuming two operation bindings or method bindings with the
//name "Commit" and "CreateInsert" exist in the binding file
//(PageDef) of the current page
OperationBinding commitOper =
  (OperationBinding) bindings.get("Commit");
OperationBinding createInsert =
  (OperationBinding) bindings.get("CreateInsert");
//execute the commit
commitOper.execute();
//check for errors
if(commitOper.getErrors().size()>0){
... //handle error
}

//execute createInsert operation
createInsert.execute();

//check for errors
if(createInsert.getErrors().size()>0){
... //handle error
```

```
}
```

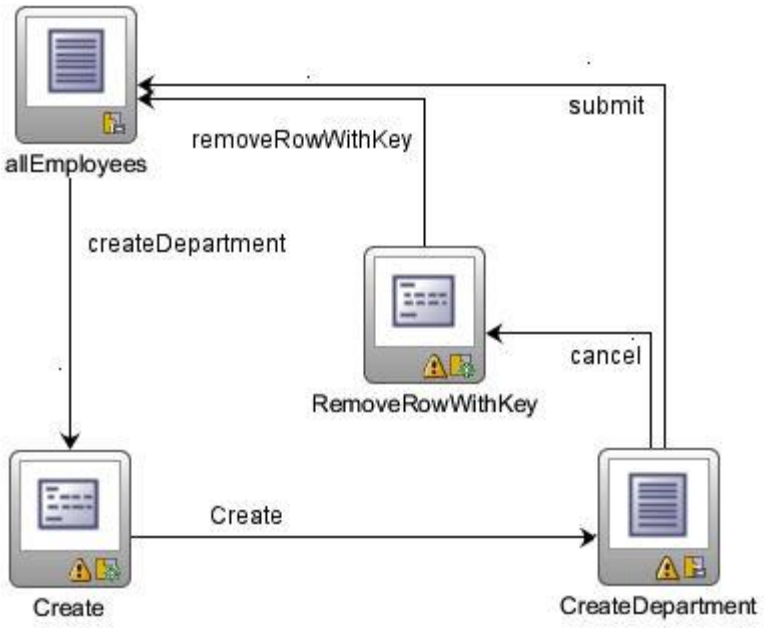
I am glad you ask. What if the operation of method requires input arguments? In this case you use code as shown below

```
OperationBinding methodWithArgs = (OperationBinding)
bindings.get("MyMethodWithArguments");
//add argument
methodWithArgs.getParamsMap().put("argument_name1",value1);
methodWithArgs.getParamsMap().put("argument_name2",value2);
//execute the methodWithArgs
methodWithArgs.execute();
//check for errors
if(commitOper.getErrors().size()>0){
 ... //handle error
}
```

Problem **You want to create an edit form for new records. When the user presses cancel, you want the new row to be deleted. The task flow (unbounded or bounded) you developed for this looks like in the image shown below.**

Starting from a browse page (allEmployees in the image) the navigation is to a Create method activity that you developed using the "CreateInsert" operation of the View Object exposed by the ADF BC Data Control. This creates and add a new line to the collection in the transaction. To be able to cancel the form, you set the "immediate" property of the cancel button to "true" and navigate to a method activity that you created by dragging the RemoveRowWithKey operation from the DataControls panel.

The RemoveRowWithKey operation expects the key string of the row to delete as an input parameter and you wonder how you can get to this information.



Solution To get the row key of the row to delete, a solution is to add an af:setPropertyListener component to the cancel button, set its type to "action" and to read the current rows key string from the ADF binding, for example: #{bindings.allDepartmentsIterator.currentRowKeyString}. Then in the target property, define a memory attribute that temporarily takes the value so it can be read by the RemoveRowWithKey method activity, for example #{requestScope.rowToDeleteKey}.

However, using ADF Business Components, the active model keeps track of the current row, which means that if the *RemoveRowWithKey* operation is from the same View Object then the edit form - which it should be anyway to delete a row from the current form - then the current row is already set. So here is another - elegant - solution to the above problem: Drag and drop the RemoveRowWithKey operation from the DataControls palette to the task flow diagram. The key argument the operation expects is referenced using EL. However, when you use the EL building in JDeveoper 11g R1 PS1 then you don't see an active binding container, though it is there at runtime. When you drag the RemoveRowWithKey operation, then implicitly a

PageDef file is created for it, which contains the operation name in the "bindings" section and the View Object iterator in its executables section. Because ADF Business Components keeps track of the current row - as mentioned before - then the row the iterator of the RemoveRowWithKey method activity's PageDef file points to the same row as the iteraor of the edit form. This means however, that you can add the following EL to the method activity key argument property: #{bindings.allDepartmentsIterator.currentRowKeyString}. Make sure you replace "allDepartmentsIterator" with the name of the iterator used in your application.

Using the above solution - and I admit that for long time even I was fooled by the EL builder in Oracle JDeveloper - you eliminate the need for af:setPropertyListener and usage of a memory attribute. All you use is the information in the binding layer, which is what is recommended as best practices anyway.

Disclaimer: Any business service that doesn't keep the model state like ADF Business Components must cintinue using the af:setPropertyListener approach mentioned before

Problem: You need generic code to find the ADF iterator binding of an ADF Faces component instance

Solution: Tree, table and tree table components reference the Collection Model, which wraps the tree binding. Input components like inputText extend the UIXEditableValue. With this information, you can use Java and EL to retrieve the ADF iterator binding - and with a few modifications - the ADF tree and attribute bindings as well.

```
//Note that the returned value is DCIteratorBinding for
//Collections and input components. Null is returned if the
//component is not tree, table, tree table or an instance of
//UIXEditableValue

private DCIteratorBinding getBinding(UIComponent comp) {

  if (comp instanceof UIXTable || comp instanceof UIXTree) {
  //get the ADF tree binding from the table or tree definition
  //and return the DCIteratorBinding
    CollectionModel model = comp instanceof UIXTable ?
      (CollectionModel)((UIXTable)comp).getValue() :
      (CollectionModel)((UIXTree)comp).getValue();
    JUCtrlHierBinding adfTreeBinding =
           (JUCtrlHierBinding)model.getWrappedData();
     return adfTreeBinding.getDCIteratorBinding();
   }

 //get iterator binding from editable value holders like
 //inputText components
 if (comp instanceof UIXEditableValue) {
 //attribute bindings always use an expression like
   bindigs.attribute.inputValue
   //to get to the attribute binding,we need to remove the
   //inputVaue reference from the EL string
   ValueExpression ve = comp.getValueExpression("value");
   if (ve.getExpressionString().indexOf(".inputValue") > -1) {
    int indx =
```

```
            ve.getExpressionString().indexOf(".inputValue");
        String attrBindingExpression =
               ve.getExpressionString().substring(0, indx) + "}";
        //get a handle to the ADF binding attribute
        FacesContext fctx = FacesContext.getCurrentInstance();
        ELContext elctx = fctx.getELContext();
        ExpressionFactory elfactory =
              fctx.getApplication().getExpressionFactory();

        ValueExpression ve2 =
            elfactory.createValueExpression
            (elctx, attrBindingExpression, Object.class);
        //cast the value expression object to a generic ADF
        //binding class
        JUCtrlAttrsBinding attributeBinding =
                     (JUCtrlAttrsBinding)ve2.getValue(elctx);
        //get iterator binding and return
        return attributeBinding.getDCIteratorBinding();
       }
      }
     return null;
    }
}
```

You call this method e.g. within a managed bean and pass the component instance, for example RichTable in.

Problem: You are not sure about how ADF binding handles concurrent requests and threads

Solution: The behavior is not really something you can configure or influence, but something worth to know:

The access to ADFm DataControls and BindingContainersis synchronized. ADFm DataControls and BindingContainers are managed by a window scope that exists within a session. A session may have multiple window scopes depending upon how many browser windows are open. DataControl and BindingContainer access are always synchronized for a given window scope. If the application only has one window open this is equivalent to synchronizing DataControl and BindingContainer access for the session.

ADF Faces Framework Solutions

Problem: You want to undo a form edit but calling AdfFacesContext.getCurrentInstance().addPartialTrigger(<component reference>) does not re-set the value

Solution: Simon Lessard posted the following hint on he OTN forum: There are several options to try

- You can call resetValue on the component since components must have local values set.
- You can perform navigation from the page to itself.
- Or, the easiest way to call reset logic would be to add the following in your returnListener
  (if the edit form is in a dialog)

```
private static final ActionListener RESET_LISTENER = new
ResetActionListener();

public void myReturnListener(ReturnEvent ev)
{
 // ...

 RESET_LISTENER.processAction(new
ActionEvent(e.getComponent()));
}
```

Problem: **How-to disable the before-session timeout message that is shown by ADF Faces starting Oracle JDeveloper 11g PS3?**

Solution The new features allows developers to specify an advanced time in which ADF Faces informs the users that his/her session is about to expire. To disable it, set the following in web.xml

```
<context-param>
  <param-name>
     oracle.adf.view.rich.sessionHandling.WARNING_BEFORE_TIMEOUT
  </param-name>
  <param-value>-1</param-value>
</context-param>
```

Disable it for unauthenticated pages in an unauthenticated page, in af:document, set stateSaving attr to client. e.g.

<af:document title="Test" id="d1" stateSaving="client">

To change the expiry time

Modify the seesion timemout or WARNING_BEFORE_TIMEOUT in web.xml
warning time = session timeout - WARNING_BEFORE_TIMEOUT

Problem: **You want to skin a component but you have no idea which selector to choose. You wonder what is the best way to find this out**

Solution: The skin selectors are documented for ADF Faces and the ADF Faces DVT components. To discover skin selectors at runtime, you do the following:

Disable content compression for the generated HTML output of the ADF Faces page. This

will change the style class names from being obfuscated, which we use to reduce the download size of pages. Astyle class that shows as .x3s at runtime might show as af_inputText_content after this. The af_inputText_content then is waht you need to discover and translate into a valid skin selector.

```
<context-param>
 <param-name>
   org.apache.myfaces.trinidad.DISABLE_CONTENT_COMPRESSION
 </param-name>
 <param-value>
     true
  </param-value>
 </context-param>
```

Run the ADF Faces application you want to discover the skin selector for in FF with Firebug and the WebDeveloper plugin installed (If you are on IE, please install FF and learn that there are some cood technologies not available in IE. However, as a developer you should have different browsers installed anyway for various reasons).

Select the tab and look at the skin selectors displayed as the style class properties. If you find someting like af_panelTabbed_tab then this translates to af|panelTabbed::tab. If you see something like .p_selected then this means its a pseudo class that needs to be appended to the selector like af|panelTabbed::tab:selected.

Use Firebug in inspect mode and select the tab. This shows you all the selectors as style classes in the generated HTML output. Open the Web Developer plugin to edit CSS on the page. Then add the style class selector you found, e.g. .af_panelTabbed_tab (note the leading dot ".") or with a curly brace

```
.af_panelTabbed_tab{ ... }
```

Within the cury braces, type the CSS you want to set and see how it behaves. You may have to play a bit to exactly find the style class and css you want, but assuming that the follwing works

```
.af_panelTabbed_tab .p_selected{background-color:red}
```

then this translates to teh following skin selection to be copied to the CSS skin file

```
af|panelTabbed::tab:selected{background-color:red}
```

Note that this backward translation from style class to skin selector needs some patience. However, experience proves to be the best when cooking and so it does for skinning.

ADF Controller Solutions

Problem: You need to generate a redirect URL for the current view activity in a bounded task flow but want to make sure the controller state for this page is properly added to the generated URL string.

Solution: The ControllerContext class exposes methods for this. The below example redirects a page to itself after reading its viewId and creating the encoded URL string

```
private void redirectToSelf(){
 FacesContext fctx = FacesContext.getCurrentInstance();
 ExternalContext ectx = fctx.getExternalContext();
 String viewId = fctx.getViewRoot().getViewId();
 ControllerContext controllerCtx = null;
 controllerCtx = ControllerContext.getInstance();
 String activityURL =
 controllerCtx.getGlobalViewActivityURL(viewId);
 try {
   ectx.redirect(activityURL);
   fctx.responseComplete();
 } catch (IOException e) {
   //Can't redirect
   e.printStackTrace();
   fctx.renderResponse();
 }
}
```

Problem You want to navigate to a page by referencing its physical file (jspx) using a URL View Activity. The challenge is to encode the URL so that all ADFc scopes managed for this page, like viewScope, are cleared.

To make sure the ADF Controller handles its scope when navigatin between application pages using a redirect to physical page files, the target URL must be created through the ADF Controller. This encoding ensures that the ADF controller context is maintained.

```
private String getPageUrl(String pagePath)
  {
     ControllerContext ctx =
           ControllerContext.getInstance();
     String url = ctx.getGlobalViewActivityURL(pagePath);
   return url;
  }

  private String getPageUrl(String pagePath)
  {
    ControllerContext ctx =
             ControllerContext.getInstance();
     String url = ctx.getGlobalViewActivityURL(pagePath);
     return url;
  }
```

```
public void setTargetPage(String targetPage)
{
   mTargetPage = targetPage;
}

public String getTargetPage()
{
   return mTargetPage;
}

public String getTargetPageUrl()
{
   return getPageUrl(getTargetPage());
}
```

The method highlighted in bold is referenced from the Url view activity. To obtain the target navigation URL, you need to call setTargetPage from a setPropertyListener added to the command component triggering the navigation. For example:

```
<af:commandButton text="goto-page1" id="cb2"
                  action="goto-dynamic-page">
  <af:setPropertyListener from="#{'pages/page1.jspx'}"
                          to="#{urlBean.targetPage}"
                        type="action"/>
```

The above page source ensures that the target URL is encoded in an ADFc compatible format.

ADF Business Components Solutions

Problem: You need to get the JDBC URL of an ADF BC bound ADF application.

Solution: The JDBC connection can be accessed from the transaction in ADF Business Components. The code below is used from the ApplicationModuleImpl method. .

```
DBTransaction transaction = this.getDBTransaction();
PreparedStatement preparedStatement =
    transaction.createPreparedStatement("commit;",0);
Connection conn;

try {
 conn = preparedStatement.getConnection();
 System.out.println(" ---- "+conn.getMetaData().getURL());
} catch (SQLException e) {
  e.printStackTrace();
}
```

Note that the commit operation is not executed. It is only used to create the statement.

☒	
☒	
☒	