ADF Code Corner

035. How-to pass values from a parent page to a popup dialog

twitter.com/adfcodecorner

Abstract:

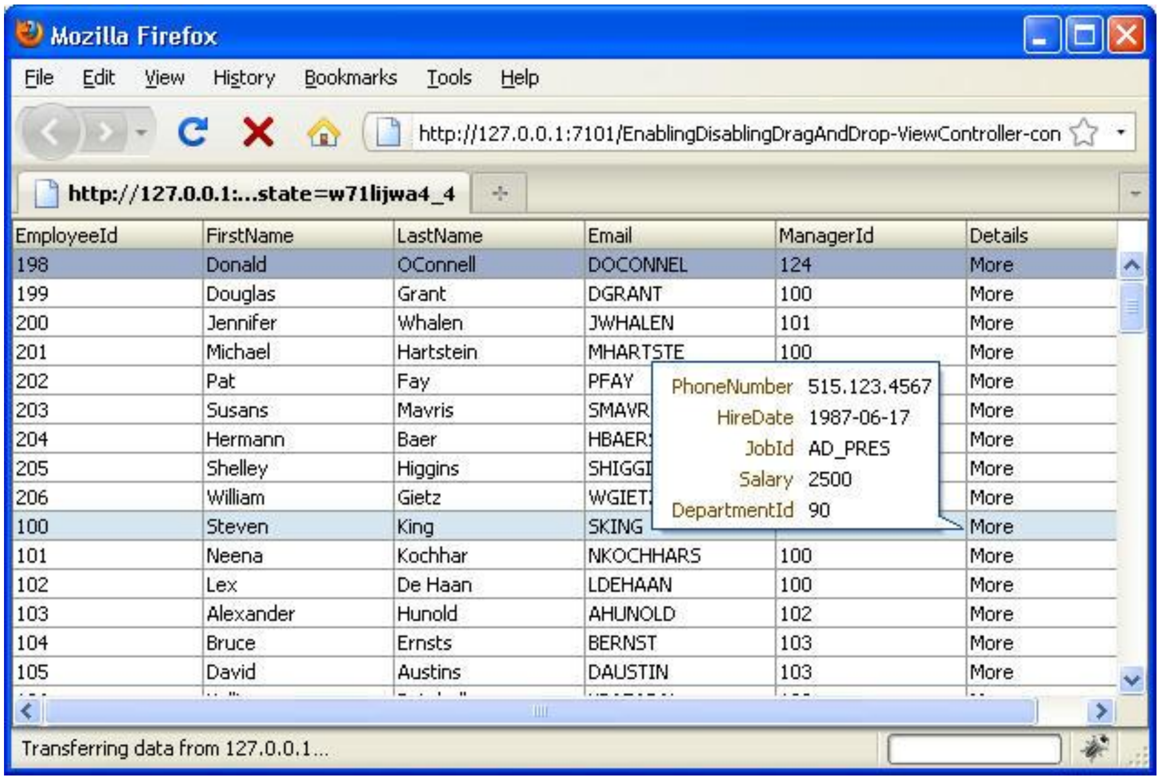
The af:popup component is used as the base component for launching dialogs and note windows. To pass information from the parent page to the popup, the easiest option is to display values of the selected, the current row. In this blog article, we explain how to pass values of a non-current row to a popup dialog so it can be used for display or in java, using a launch listener.

Author: Frank Nimphius, Oracle Corporation
twitter.com/fnimphiu
02-NOV-2009

Introduction

To display values of the current table row in a popup dialog, or read current row data values in Java, developers use a simple but effective binding trick: To access the values of a selected row, you create attribute bindings for all attributes of interest of the iterator that the tree binding, which populates the tree at runtime, is based on. Because tables synchronize the selected table row with the binding, the attribute bindings always provide access to the row values using #{bindings.<attribute name>.inputValue}. This little trick comes with no additional cost and also has the advantage that you can use the Expression Language builder in Oracle JDeveloper to bind UI components to the values. The disadvantage of this approach is that a row must be current - selected - for the popup to display the correct values or Java to be able to read these values in. Use cases, like multi row selection or displaying detail information fro a row, require a special handling if the information should be for non-current rows as well. In the example explained in this article, we use a table that displays a note window when the mouse hovers over a text label in the last column. The implementation code can be easily changed by you to launch the dialog from an image or one of the existing cell information.

In the image below, hidden columns of an af:table component are displayed in a note window when moving the mouse over the "More" text string in the Details column. As mentioned before, the implementation code also allows you to modify it such that the the popup is displayed when moving the mouse over an image or one of the displayed columns. Note in the image below that the first table row is the selected row, whereas the information displayed in the popup is from a non-current row.


Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://127.0.0.1:7101/EnablingDisablingDragAndDrop-ViewController-con
http://127.0.0.1:...state=w71lijwa4_4

EmployeeId	FirstName	LastName	Email	ManagerId	Details
198	Donald	OConnell	DOCONNEL	124	More
199	Douglas	Grant	DGRANT	100	More
200	Jennifer	Whalen	JWHALEN	101	More
201	Michael	Hartstein	MHARTSTE	100	More
202	Pat	Fay	PFAY		More
203	Susans	Mavris	SMAVR		More
204	Hermann	Baer	HBAER		More
205	Shelley	Higgins	SHIGGI		More
206	William	Gietz	WGIET		More
100	Steven	King	SKING		More
101	Neena	Kochhar	NKOCHHARS	100	More
102	Lex	De Haan	LDEHAAN	100	More
103	Alexander	Hunold	AHUNOLD	102	More
104	Bruce	Ernsts	BERNST	103	More
105	David	Austins	DAUSTIN	103	More

PhoneNumber	515.123.4567
HireDate	1987-06-17
JobId	AD_PRES
Salary	2500
DepartmentId	90

Transferring data from 127.0.0.1...

To implement this solution, you need to do as follows

- Create a popup that loads its content lazily but uncached
- Define the popup launch context to be the context of the launcher component
- Create client attributes to the launcher component that contain the information you want to pass to the dialog
- Use setPropertyListeners to store this information in an object in memory when the popup launches
- Read the information from the memory object

Building the table

The table component is created by dragging a view object Collection from the DataControls palette to the JSF page. Choose all columns that you want to display as table columns, or that you want to show information about in the popup. This creates a tree binding with the selected attribute values. Use the Structure window to delete the column definitions you don't want to display in the page. For this, expand the af:table in the Structure window and select the columns to remove. Then hit delete, to remove the column page sources, preserving the ADF binding definition for these attribute. In the Structure window, add a new af:column to the table and add an af:outputText component, an image, a link or whatever component you like to render the Detail cell. In this article we use an af:outputText component.

Creating the popup dialog

The popup dialog is built by dragging the Popup component from the ADF Faces component palette onto the JSF page and adding a Note Window component as its child. Importantly, the ContentDelivery property of the af:popup component is set to lazilyUncached to ensure the popup content is renewed when the popup opens.

You need to define a value for the LauncherVar property and set the EventContext to "launcher". This way it is possible to access attributes, including custom clientAttributes, from the launching component within the context of the opened popup dialog. Accessing the launcher component attributes allows us to pass the row data information from the Detail cell to the popup.

```
<af:popup id="p1" launcherVar="source" contentDelivery="lazyUncached"
          eventContext="launcher">
   <af:noteWindow id="nw1">
     <af:panelFormLayout id="pfl1">
       <af:panelLabelAndMessage label="PhoneNumber" id="plam1">
         <af:outputText value="#{viewScope.phone}" id="ot5"/>
       </af:panelLabelAndMessage>
        <af:panelLabelAndMessage label="HireDate" id="plam2">
       <af:outputText value="#{viewScope.hireDate}" id="outputText1"/>
       </af:panelLabelAndMessage>
       <af:panelLabelAndMessage label="JobId" id="plam3">
         <af:outputText value="#{viewScope.jobId}" id="outputText2"/>
       </af:panelLabelAndMessage>
       <af:panelLabelAndMessage label="Salary" id="plam4">
         <af:outputText value="#{viewScope.salary}" id="outputText3"/>
       </af:panelLabelAndMessage>
       <af:panelLabelAndMessage label="DepartmentId" id="plam5">
         <af:outputText value="#{viewScope.departmentId}"
                        id="outputText4"/>
       </af:panelLabelAndMessage>
     </af:panelFormLayout>
   </af:noteWindow>
   <af:setPropertyListener from="#{source.attributes.phone}"
                           to="#{viewScope.phone}" type="popupFetch"/>
   <af:setPropertyListener from="#{source.attributes.hireDate}"
                           to="#{viewScope.hireDate}"
                           type="popupFetch"/>
   <af:setPropertyListener from="#{source.attributes.jobId}"
                           to="#{viewScope.jobId}" type="popupFetch"/>
   <af:setPropertyListener from="#{source.attributes.salary}"
                           to="#{viewScope.salary}" type="popupFetch"/>
   <af:setPropertyListener from="#{source.attributes.departmentId}"
                           to="#{viewScope.departmentId}"
                           type="popupFetch"/>
</af:popup>
```

In the page source above, which defines the popup dialog, a af:panelLabelMessage component is used to create the read only form that is displayed in the popup dialog. The af:setPropertyListener at the end of the dialog definition is used to read attribute values that are defined on the the launcher component (an output text component in this example). The values are written to the viewScope, which ensures they are cleaned up when the user navigates away from the current JSF page. The viewScope attributes are accessed - or referenced - from the outputText components that display the values in the noteWindow.

Launching the Popup

To launch the pop, we use an af:showPopupBehavior component on the af:outputText component that renders the "Details" column cell.

```
<af:column id="c2" headerText="Details">
  <af:outputText value="More" id="ot1" clientComponent="true">
     <af:clientAttribute name="phone" value="#{row.PhoneNumber}"/>
     <af:clientAttribute name="hireDate" value="#{row.HireDate}"/>
     <af:clientAttribute name="jobId" value="#{row.JobId}"/>
     <af:clientAttribute name="salary" value="#{row.Salary}"/>
     <af:clientAttribute name="departmentId"
                         value="#{row.DepartmentId}"/>
     <af:showPopupBehavior popupId="::p1" triggerType="mouseHover"
                           align="endAfter" alignId="ot1"/>
  </af:outputText>
</af:column>
```

The af:clientAttribute tags are used to add the row attribute values that should be displayed in the af:popup to the af:outputText component. Note that the "row" variable belongs to the table and is only available when the table renders. This is why we need to store the row value information as a custom attribute with the launcher component. The af:showPopupBehavior component then launches the popup relative to the outputText component when the mouse hovers over the text field. Note that you could also use mouseOver as the trigger to launch the popup. The difference between mouseOver and mouseHover is a slight delay that is enforced using mouseHover. This way we avoid popups to be launched only because a nervous application user moves the mouse too quickly over the launcher component.

Download the sample

The sample workspace is built with Oracle JDeveloper 11g R1 and connects to a local database with the HR schema installed and enabled. You need to change this connection so it matches your database con next details. Run the JSPX page and move the mouse over the "More" string in the Details column: Download the sample from ADF Code Corner: http://www.oracle.com/technetwork/developer-tools/adf/learnmore/index-101235.html

RELATED DOCOMENTATION

☒	af.popup tag documentation - http://download.oracle.com/docs/cd/E12839_01/apirefs.1111/e12419/tagdoc/af_popup.html
☒	Af:showPopupBehavior tag documentation -- http://download.oracle.com/docs/cd/E12839_01/apirefs.1111/e12419/tagdoc/af_showPopupBehavior.html
☒	