ADF Code Corner

028. How-to scroll an ADF bound ADF Faces Table using a Keyboard Shortcut

twitter.com/adfcodecorner

Abstract:

The Oracle JDeveloper code editor allows developers to navigate to a specific line in the source code using the ctrl+g keyboard shortcut. In this how-to article, I use the same approach, which is to allow users to press ctrl+g on a table at runtime to provide the row number they want to navigate to. The sample is an improved version of the sample Lynn Munsinger and I provide in our book "Oracle Fusion Developer Guide Building Rich Internet Applications with Oracle ADF Business Components and ADF Faces" and nicely shows how JavaScript can be used to provide client side functionality in ADF Faces RC, plus how the ADF binding layer is used to scroll a table

Author: Frank Nimphius, Oracle Corporation
twitter.com/fnimphiu
08-JAN-2010

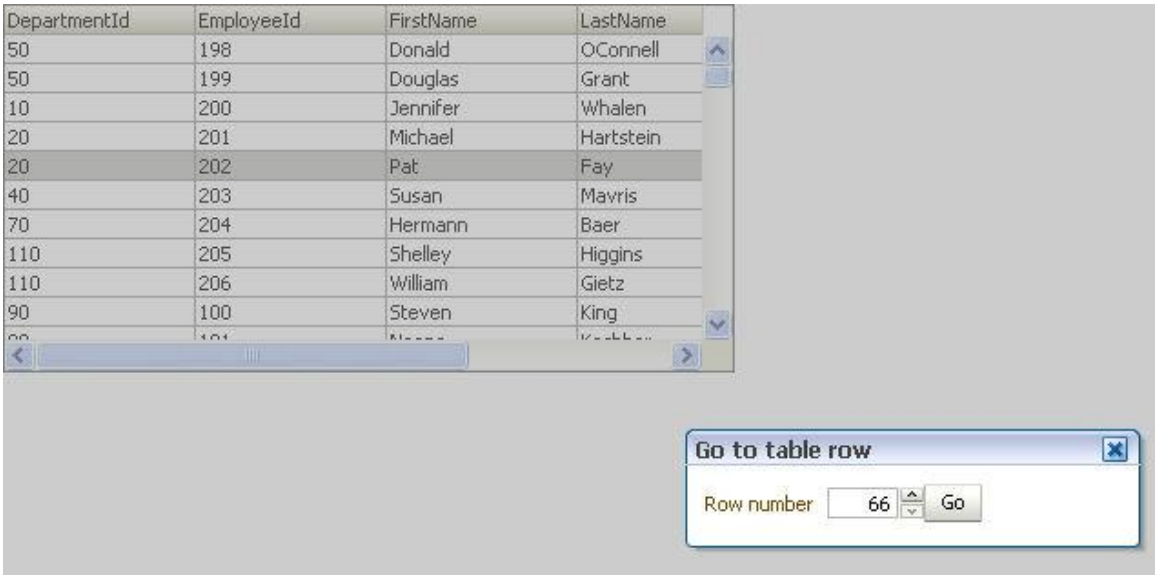
Oracle ADF Code Corner is a loose blog-style series of how-to documents that provide solutions to real world coding problems.

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.

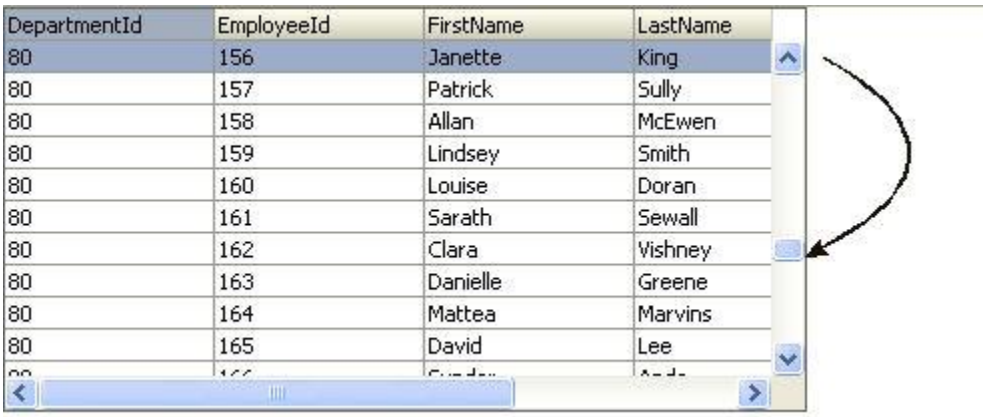
Please post questions or report problems related to the samples in this series on the OTN forum for Oracle JDeveloper: http://forums.oracle.com/forums/forum.jspa?forumID=83

Introduction

In a previous article, I showed how a character index menu can be build and used to scroll an ADF bound table. Clicking an index link, users navigate and select the first occurrence of a row matching the search criteria represented by the link [see]. In this article, I demonstrate how users can scroll a table to a specific table row knowing about the row number they want to access. Pressing the ctrl+g keyboard shortcut, with the table selected, opens a dialog that users type in the row number to access.



Pressing the "Go" button navigates the table as shown below



The af:table component has its display row property set to "selected" to always show the selected row on top of the table, which is why the row always shows as the first.

View Layer Implementation

The implementation of this solution is view layer only, using JavaScript and a managed bean. The way it works is that a JavaScrip event listener is assigned to the ADF Faces table using the af:client listener. The called JavaScript function looks for the pressed keys and - if detecting ctrl+g - opens an af:popup dialog for the user to provide the row number.

JavaScript

The JavaScript code can be added somewhere within the af:document tag using the af:resource tags. The af:resource tag ensures that the JavaScript code uses an optimized download to the client and for this reason is recommended to use.

```
<af:resource type="javascript">
 function launchTableRowScroller(evt){
 G_KEY = 71;
 //call delete command if ctrl+g key is pressed
 keyPressed = evt.getKeyCode();
 modifiers = evt.getKeyModifiers();
 if (keyPressed == G_KEY){
   if(modifiers == AdfKeyStroke.CTRL_MASK){
     evt.cancel();
     popup = AdfPage.PAGE.findComponentByAbsoluteId('p1');
     var hints = {};
     popup.show(hints);
   }
  }
 }
</af:resource>
```

The JavaScript function gets the key code from the event object that is passed in to the JavaScript function. The event object is an ADF Faces keyboard event that allows us to read the pressed key and the

modifier (if one is pressed). This information then is compared to the "g" character code, which is 71. If ctrl+g is pressed, then a popup dialog that is defined on the same page is launched and the keyboard event is canceled so it doesn't continue bubbling. If the event is canceled first (like the book example does) then all keyboard events on the table are canceled, which is a technique you can use to suppress any key to be handled by the browser.

The JavaScript function is configured on the af:table component as shown below:

```
<af:table value="#{bindings.EmployeesView1.collectionModel}"
          var="row" rows="#{bindings.EmployeesView1.rangeSize}"
 ...>
   <af:clientListener type="keyDown" method="launchTableRowScroller"/>
</af:table>
```

Whenever a keyboard down event is performed on the table, the JavaScript function us called. Note that the function name is used only, The event object is implicitly passed in to the function.

af:popup

The popup dialog is defined as a DHTML popup window in ADF Faces.

```
<af:popup id="p1">
 <af:panelWindow id="pw1" modal="true" inlineStyle="width:250px;"
                 title="Go to table row">
   <af:panelGroupLayout id="pgl1" layout="horizontal">
     <af:inputNumberSpinbox label="Row number" id="ins1" minimum="1"
              maximum="#{bindings.EmployeesView1.estimatedRowCount}"
             value="#{SearchRowBean.rowNumber}"/>
        <af:commandButton text="Go" id="cb1" partialSubmit="true"
                          actionListener="#{SearchRowBean.onRowSearch}"/>
   </af:panelGroupLayout>
  </af:panelWindow>
</af:popup>
```

Pressing the "Go" button in the dialog calls a managed bean method that gets the rowKey of the requested row to then set it as the current row in the underlying ADF binding. The af:inputNumberSpinbox component has its value property set to reference a managed bean property. This way the user provided row number is accessible in the managed bean.

Managed Bean Code

The major part of the job is done by the managed bean, which I configured in request scope because its maintains the state of the number input component.

```
public class SearchRowBean {

 Integer rowNumber = 1;
 private RichTable table1;

 public SearchRowBean() {
 }

 public void onRowSearch(ActionEvent actionEvent) {
```

```
  //get the table model
  CollectionModel collectionModel =
                   (CollectionModel)table1.getValue();
  //the table model - CollectionModel - wraps the ADF tree
  //binding for this table
  JUCtrlHierBinding tableBinding =
        (JUCtrlHierBinding) collectionModel.getWrappedData();
  //get the iterator for the tree binding
  DCIteratorBinding iteratorBinding =
                            tableBinding.getIteratorBinding();
  //from the table instance itself, get the requested row number
  JUCtrlHierNodeBinding rowBinding =
           (JUCtrlHierNodeBinding) table1.getRowData(rowNumber-1);
  //from the node binding, get the JBO rowKey
  Key rowKey = rowBinding.getRow().getKey();
  //make the searched row the current
  iteratorBinding.setCurrentRowWithKey(rowKey.toStringFormat(true));
  //create a new table rowKey (the RichTable row key
  //is different from JBO Key
  ArrayList tableRowKey = new ArrayList();
  tableRowKey.add(rowKey);
  RowKeySetImpl rks = new RowKeySetImpl();
  rks.add(tableRowKey);
  table1.setSelectedRowKeys(rks);
  //close the search dialog
  closePopup("p1");
  //refresh table
  AdfFacesContext.getCurrentInstance().addPartialTarget(table1);
}

private void closePopup(String popup) {
  FacesContext fctx = FacesContext.getCurrentInstance();
  //create the JaavScript expressions
  StringBuffer scriptBuffer = new StringBuffer();
  scriptBuffer.append(
  "var popup = AdfPage.PAGE.findComponentByAbsoluteId('");
  scriptBuffer.append(popup+"');");
  scriptBuffer.append("if(popup.isPopupVisible()==true){");
  scriptBuffer.append("popup.hide();}");
  String script = scriptBuffer.toString();
  //execute the script on the client
  ExtendedRenderKitService extendedRenderKitService =
  Service.getRenderKitService(fctx,ExtendedRenderKitService.class);
  extendedRenderKitService.addScript(fctx,script);
}

public void setRowNumber(Integer rowNumber) {
  this.rowNumber = rowNumber;
}

public Integer getRowNumber() {
  return rowNumber;
}

public void setTable1(RichTable table1) {
  this.table1 = table1;
```

```
  }

  public RichTable getTable1() {
    return table1;
  }
}
```

As you can see by reading the code, the managed bean also is responsible for closing the popup dialog before refreshing the table. This sample therefore also provides yo with a demo of how to call client side JavaScript from Java in a managed bean.

Note: As of Jdeveloper 11g R1 PS2 (11.1.1.3) there exist an API on the RichPopup instance to close a popup. So the above JavaScript, though still working, is not needed anymore

I commented the source code so you know what it is doing. Worth pointing out though that the table binding, the ADF tree ninding and the iterator binding are read from the table and ot looked up on the binding layer using the "bindings" object. Not accessing the "bindings" object for this means that the developers doesn't make the code dependent on the implementation he / she builds, which means that code can be reused easily.

Download Sample Workspace

The sample was built with JDeveloper 11g R1 PS1 and runs against the HR database schema of the Oracle XE and Oracle RDBMS database. You only need to change the database connection to point to your database SID and provide the HR username and password. **Get the workspace from ADF Code Corner:**

http://www.oracle.com/technetwork/developer-tools/adf/learnmore/index-101235.html

RELATED DOCOMENTATION

☒	Oracle Fusion Developer Guide – McGraw Hill Oracle Press, Frank Nimphius, Lynn Munsinger http://www.mhprofessional.com/product.php?cat=112&isbn=0071622543
☒	
☒	