ADF Code Corner

012. How-to copy/paste the value of a table cell to other - selected - table rows

twitter.com/adfcodecorner

Abstract:

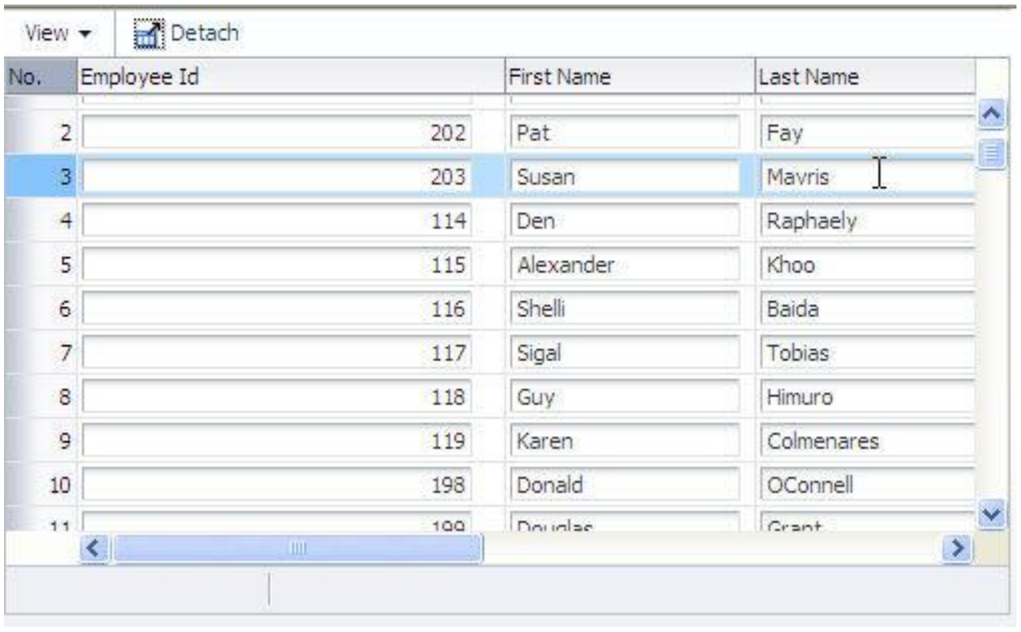
This blog article describes the solution to a very specific customer problem and requirement (aka. use case) to copy the value of a table cell to the same column in multiple selected table rows. The idea behind this use case is to enable application users to bulk update status information by copying an existing status information. The problem description is a good example for a client side JavaScript solution as there is no native API in ADF Faces tables to detect the selected table cell. In addition, though the sample works perfectly fine with ADF Business Components models, the solution is implemented with a POJO model to mimic the customer environment and to increase the amount of samples Oracle provides in this area.

Author: Frank Nimphius, Oracle Corporation
twitter.com/fnimphiu
01-JUN-2010

Introduction

The use case covered in this ADF Code Corner article is best explained by a series of screen shots showing the final runtime beavior. The Oracle JDeveloper 11.1.1.3 workspace for this example is provided for download at the end of this article.

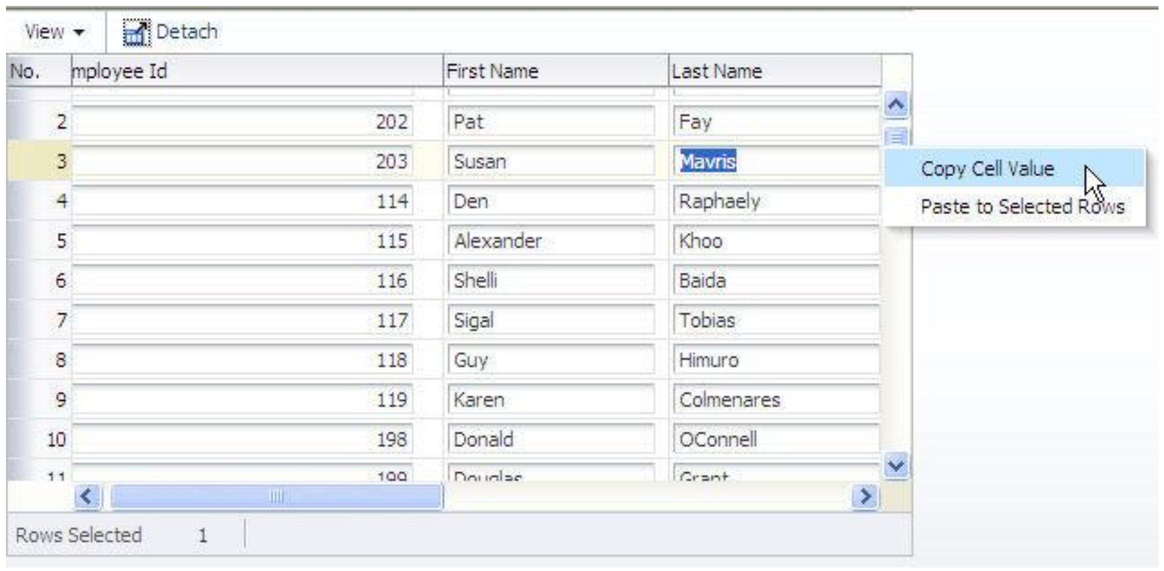
1. To bulk update selected table rows, the application user selects an editable table cell to copy the value from



No.	Employee Id	First Name	Last Name
2	202	Pat	Fay
3	203	Susan	Mavris
4	114	Den	Raphaely
5	115	Alexander	Khoo
6	116	Shelli	Baida
7	117	Sigal	Tobias
8	118	Guy	Himuro
9	119	Karen	Colmenares
10	198	Donald	OConnell
11	199	Douglas	Grant

2. With the right mouse button pressed, a context menu opens to copy the cell value. The context menu is defined using ADF Faces components and JavaScript. It cancels the context menu action to suppress the browser native functionality. The table cell renderer components are af:inputText components with an af:clientListener defined to open the menu.

Hint: Alternatively to using the context menu, the user can press ctrl+shift+c to copy the cell content. The implementation for this is through another af:clientListener that looks at the keyboard code and the key modifiers pressed using the ADF Faces client side APIs.

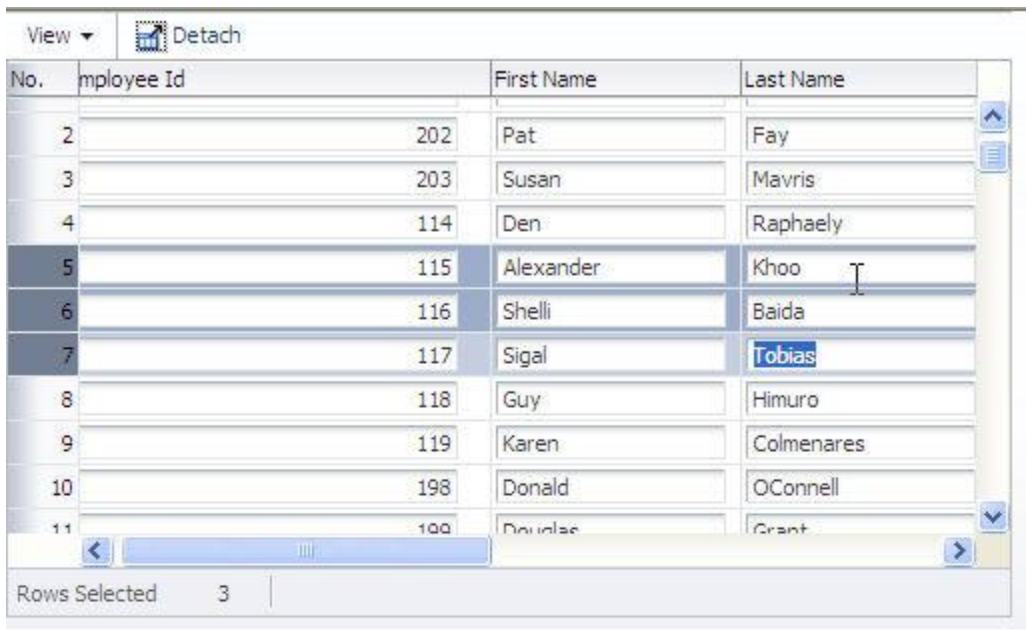


No.	mployee Id	First Name	Last Name
2	202	Pat	Fay
3	203	Susan	Mavris
4	114	Den	Raphaely
5	115	Alexander	Khoo
6	116	Shelli	Baida
7	117	Sigal	Tobias
8	118	Guy	Himuro
9	119	Karen	Colmenares
10	198	Donald	OConnell
11	199	Douglas	Grant

Rows Selected 1

3. Now the user can ress the ctrl key and click on all the rows rows he or she want to copy the values to. The implementation is such that when copying the cell value, the column is remembered as well, so that when pasting it to other rows, values are added to the right column.

Hint: The functionality can be easily extended to copy and paste the values of multiple columns. All you need to do on the page source is to add additional column names (atributes) to the colname af:clientAttribute tag, which we explore in a minute

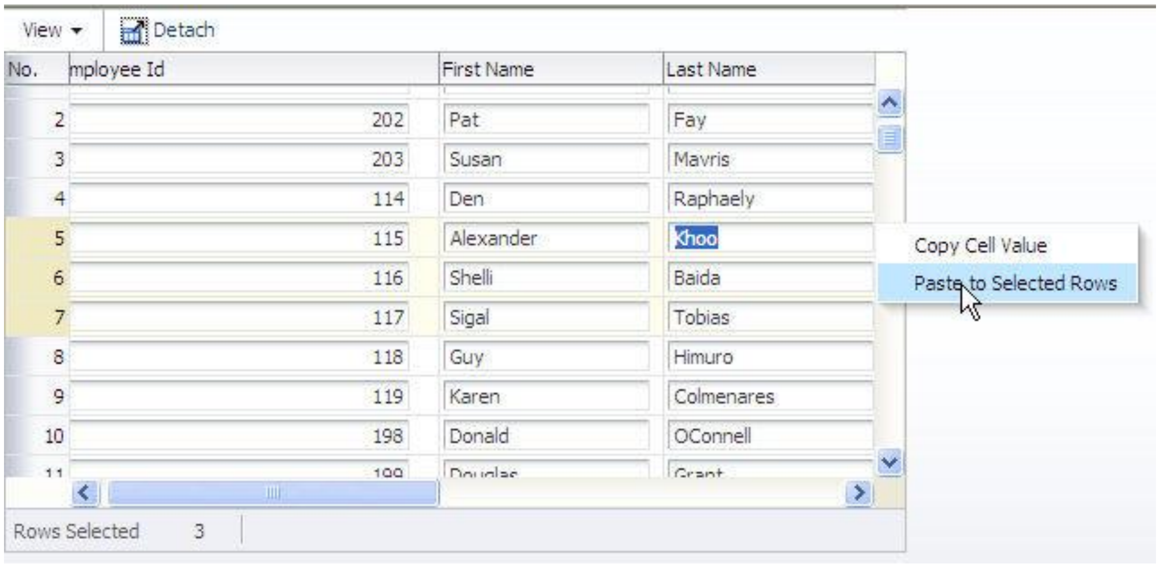


No.	mployee Id	First Name	Last Name
2	202	Pat	Fay
3	203	Susan	Mavris
4	114	Den	Raphaely
5	115	Alexander	Khoo
6	116	Shelli	Baida
7	117	Sigal	Tobias
8	118	Guy	Himuro
9	119	Karen	Colmenares
10	198	Donald	OConnell
11	199	Douglas	Grant

Rows Selected 3

4. To paste the values, the user opens the context menu with the right mouse button and chooses the paste option.

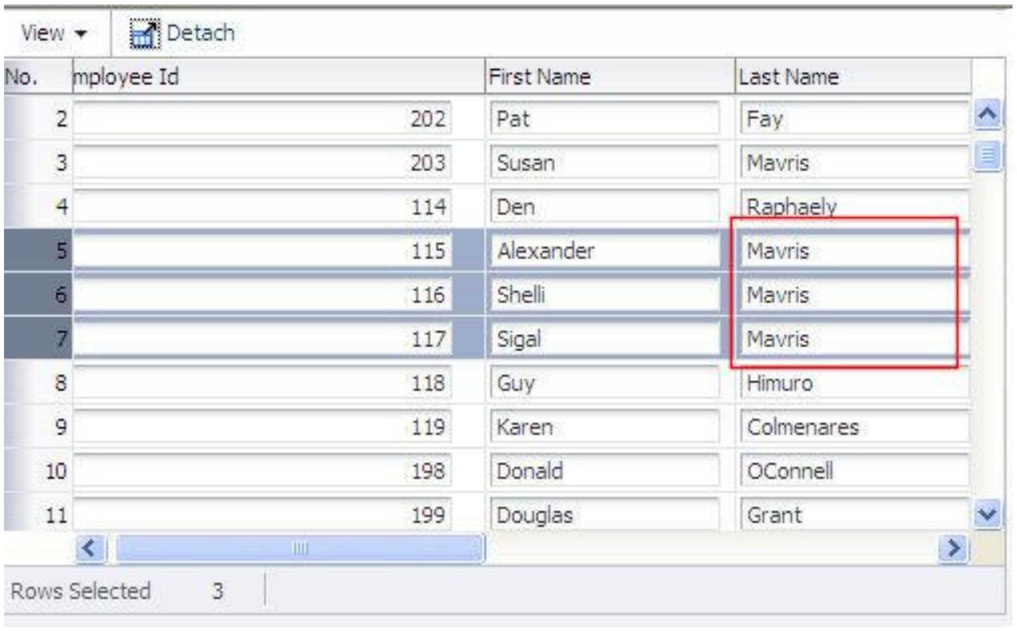
Hint: Instead of the context menu, the user could use the ctrl+shift+v keyboard combination to paste the value to the selected table rows.



No.	mployee Id	First Name	Last Name
2	202	Pat	Fay
3	203	Susan	Mavris
4	114	Den	Raphaely
5	115	Alexander	Khoo
6	116	Shelli	Baida
7	117	Sigal	Tobias
8	118	Guy	Himuro
9	119	Karen	Colmenares
10	198	Donald	OConnell

Rows Selected 3

5. As shown in the image below, the cell value is copied to the selected table rows. But there is more than what meets the eyes. The logic of copying the data is implemented on the server side, with the help of an **af:serverListener** component. The server side implementation updates the ADF binding layer for the selected rows and - optionally - can be used to update the business service as well, which especially in non-ADF Business Components cases require an extra function to be called.



No.	mployee Id	First Name	Last Name
2	202	Pat	Fay
3	203	Susan	Mavris
4	114	Den	Raphaely
5	115	Alexander	Mavris
6	116	Shelli	Mavris
7	117	Sigal	Mavris
8	118	Guy	Himuro
9	119	Karen	Colmenares
10	198	Donald	OConnell
11	199	Douglas	Grant

Rows Selected 3

At the end, the table is PPR'ed to show the copied values in the selected table rows. Of course, the implementation could have been client side only, but this would go with a lot more JavaScript coding. Because tables in ADF Faces are stamped its in fact easier to perform the value operations on teh server and accept the little twinkle when refreshing the table.

Note:Though JavaScript is mainstream with Ajax, I recommend to use it by exception and only in small doses. In traditional web developments, JavaScript has proven as a silver bullet. However, if you want to play it safe, prefer mature, solid and typed Java programming interfaces whenever you can to ensure browser and upgrade compatibility for your applications.

Here's a list of topics you learn about if you continue reading this article:

- af:clientListener, af:serverListener usage
- how to detect which table a user selected
- using JavaScript callbacks in af:clientListener to pass additional arguments
- how to create keyboard shortcuts (ctrl+shift+c and ctrl+shift+v), optionally overriding the browser defaults
- The use of DCDataRow vs. oracle.jbo.Row

Note: In a previous article, I explained pagination in POJO based DataControls. This sample too uses pagination to not fetch all data at once.

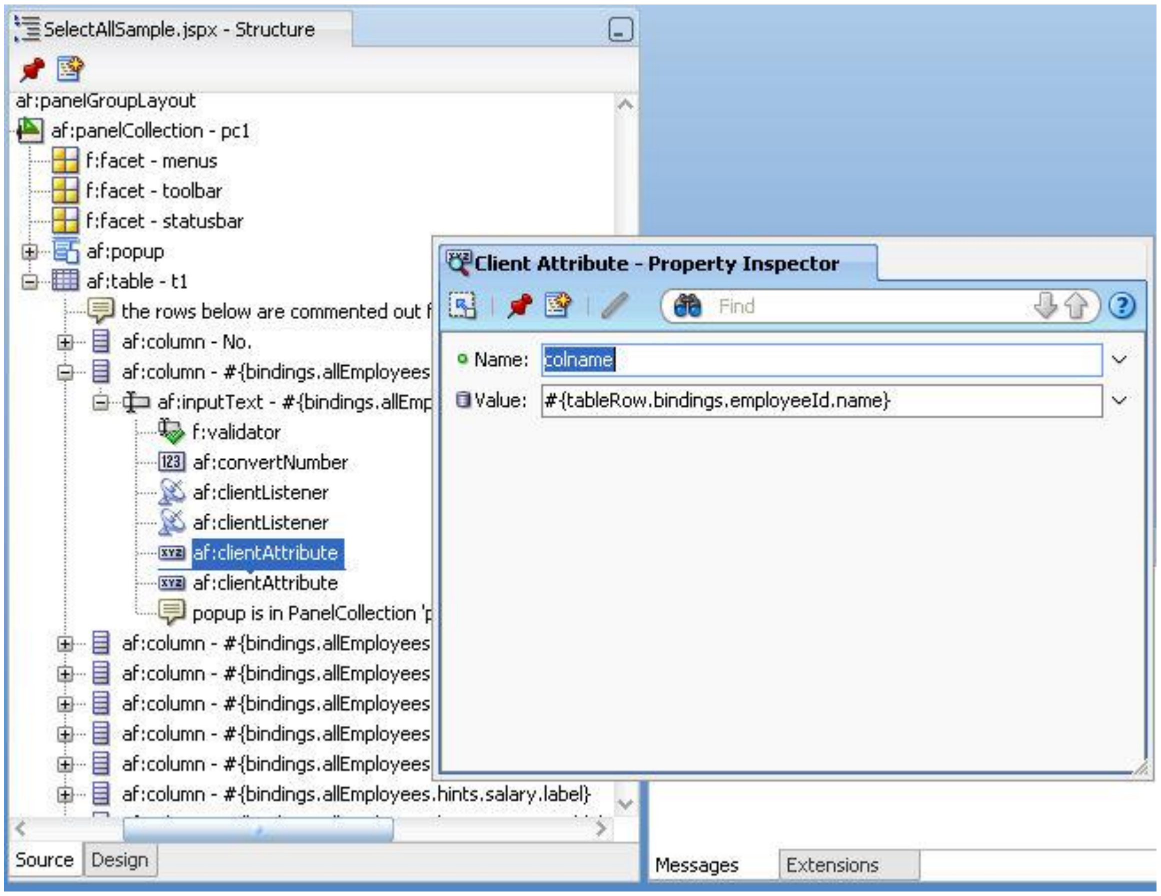
Note: I added comments to the source code printed in this article and contained in the sample workspace. Make sure you read the comments for a better understanding of what is going on.

Setting-up the table

Users select a table cell to copy the value from. The ADF Faces table is stamped, which means that a cell is not an object that one can ask for what column and table row it belongs to. So to get this information at runtime, a little trick using an **af:clientAttribute** is needed to provide this information at design time . The af:clientAttribute component allows developers to enhance ADF Faces component instences with additional - custom defined - properties. The table in the example has two af:clientAttribute tags, one for the column name (the attribute name it represents, not the column label) and one for the row number (the row index).

As you see in the image below, the client attributes are defined as a child of the **af:inputText** components that make the cell renderer. The client attribute for the column name is called **"colname"** (a name I came up with) and references the binding layer through the "**tableRow**" variable. The "tableRow" variable is defined on the table and is used to temporarily hold the current rendered table row when iterating over the CollectionModel to print the table. By default the variable name is set to **"row"**. I changed it to "tableRow" to make it more explicit what the role of this is.

The "colname" value is read from the ADF binding layer, through the tableRow variable. By accessing the binding layer for the column name, I make the page sources independent from the business service accessed by the binding. On the client side, from JavaScript, the client attribute value can be accessed by <component>.getProperty('colname').



Name: colname
Value: #{tableRow.bindings.employeeId.name}

The client listeners on the Input Text components are used to respond to the right mouse button click and the keyboard. When the user clicks on a text field using the right mouse button, a popup dialog is opened and aligned with the selected table cell. You can't do this alignment with the **af:showPopupBehavior** tag, which is why a pure JavaScript approach is taken. The keyboard listener calls a JavaScript function for each key pressed. The JavaScript function then checks if the pressed key is "c" or "v" in combination with ctrl+shift. If any other keyboard key or key combination is pressed, the JavaScript function doesn't do anything and lets the event bubble up for the browser to handle. If the key **ctrl+shift+c** or **ctrl+shift+v** is detected, then the JavaScript function cancels the event and copies or pastes the cell value.

Important: For ADF Faces components to be come accessible in JavaScript on the client, the **clientComponent** property must be set to "true" or an af:clientListener tag must be added. Otherwise, most likely, the component renders in HTML in which case it is "invisible" for the client side ADF Faces framework. But wait ! Before you set clientComponent="true" for all components on a page, keep in mind that the decision not to generate JavaScript objects for all components on a page is by design and for better performance. So please, curve your enthusiasm and think twice before enabling this option. However, forgetting to set clientComponent to true is the most common mistake developers do when using the client side AD Faces JavaScript framework.

Below is the page source of a table column, which is configured for copy and paste support. The interesting code lines are highlighted in bold. Please notice the use of the "status" variable that is the value of the table varStatus attribute. The "status" variable is used to display row numbers in front of the table but also as the value of the "rowKey" client attribute to identify the row a selected table cell component is in.

```
<af:table value="#{bindings.allEmployees.collectionModel}"
     var="tableRow" rowBandingInterval="0" id="t1"
     binding="#{EmployeesPage.employeeTable1}"
     rowSelection="multiple"
     rows="#{bindings.allEmployees.rangeSize}"
     emptyText="#{bindings.allEmployees.viewable ?
                          'No data to display.' : 'Access Denied.'}"
     fetchSize="#{bindings.allEmployees.rangeSize}"
     varStatus="status"
     contentDelivery="whenAvailable" editingMode="editAll">

    <!-- the rows below are commented out from the table definition to allow multiple
         table row selection. The ADF binding layer can only have a single current
         row and therefore would reset multiple row selection on table refresh.
         selectedRowKeys= "#{bindings.allEmployees.collectionModel.selectedRow}"
         selectionListener= "#{bindings.allEmployees.collectionModel.makeCurrent}"
   -->
  <af:column id="c5" width="31" headerText="No." rowHeader="true">
    <af:outputText value="#{status.index}" id="ot1"/>
  </af:column>
  <af:column sortProperty="employeeId" sortable="false"
        headerText="#{bindings.allEmployees.hints.employeeId.label}"
        id="c1" width="209">
    <af:inputText value="#{tableRow.bindings.employeeId.inputValue}"
       columns="#{bindings.allEmployees.hints.employeeId.displayWidth}"
          shortDesc="#{bindings.allEmployees.hints.employeeId.tooltip}"
         id="it1" clientComponent="true"
                  autoSubmit="false">
     <f:validator binding="#{tableRow.bindings.employeeId.validator}"/>
          <af:convertNumber groupingUsed="false"
          pattern="#{bindings.allEmployees.hints.employeeId.format}"/>
       <af:clientListener method="openPopup('pc1:p1')"
                              type="contextMenu"/>
      <af:clientListener
                           method="copyFromKeyboard('ServerCopy','pc1:p1')"
                           type="keyDown"/>
        <af:clientAttribute name="colname"
                             value="#{tableRow.bindings.employeeId.name}"/>
       <af:clientAttribute name="rwKeyIndx" value="#{status.index}"/>
         <!-- popup is in PanelCollection 'pc1' with an ID of 'p1' -->
   </af:inputText>
 </af:column>
...
```

With this table configuration, a JavaScript function is called when the right mouse button is used on a table cell and when users press a keyboard key. The client attributes provide information about the "where in the table" users clicked into. Note that the client listener methods have string arguments that define the name of the af:serverListener component to call a managed bean method and the client id of the popup component to launch.

JavaScript Handlers

At a first glance, the JavaScript below looks like a lot of code to write for a little functionality. However, as you will see when I go through it, most of what you see are comments I added for you to get a better understanding of what a specific code line is doing.

The JavaScript source is referenced from the ADF Faces page using the **af:resource** component, as shown below

<af:resource type="javascript" source="/jshelper.js"/>

"The resource is added to the af:document component to be included when the document is rendered. The resource is only processed on the initial creation of the document. Only JSP tags may be used in the body, no JSF components are supported, or deferred EL expressions as the value is processed during tag execution, not during component tree rendering. If the source attribute is not given, the content of the tag is used for in-line CSS style or JavaScript, otherwise the source attribute is used as the URI to the resource. af:resource must be a descendent of af:document." --- (from the tag documentation)

The JavaScript has two global variable defined, which are used as a temporary memory scope for the selected table cell content. The **globalFieldCopy** variable holds the inputText component instance of the table cell to copy from. The **globalLastVisitedField** variable keeps track of the last visited table cell component. This variable is copied into the **globalFieldCopy** variable whenever the user selected context menu action is COPY.

The **openPopup(popupId)** function uses a JavaScript callback mechanism to receive a user defined argument, the id of the popup to lookup and open, as well as the ADF Faces event object. JavaScript callbacks are a great help when the goal is to write reusable code that could be saved in a JavaScript library. Similar to ADFUtils and ADFFacesUtils, the two helper classes used within Oracle sample applications like Fusion Order Demo (FOD) , you could build a JSUtils as well. The popup is opened in response to a context menu event, which also allows access to the table cell component. This component reference is copied into the **globalLastVisitedField** variable. The popup then reads the *clientId* of the table component, which is the absolute component locator id (kind of the address a component has in the rendered table), to then provide this as the alignId for the context menu to open next to the selected table cell.

The **copyFromKeyboard** method handle the **ctrl+shift+c and ctrl+shift+v** keyboard short cuts to copy and paste the selected cell value. As an additional argument, it needs the id of the af:serverListener and the component Id of the component that owns the server listener. Interesting for you to read in this function is the way the ADF Faces client side framework supports developers to detect keyboard strokes and the pressed keyboard modifiers through its AdfKeyStroke class..

The **copyFromMenu** function is called when a user clicks the "Copy" menu option and copies the value held by the globalLastVisitedField into the globalFieldCopy variable.

The **pasteFromMenu** function is called when the user selects the "Paste" option from the context menu and calls, like copyFromKeyboard in the "Paste" case, calls the **copyValueToSelectedRows** function.

The **copyValueToSelectedRows** function is called from both "Paste" functions (the menu and the keyboard) and uses the **af:serverListener** to queue a custom event to the server. The information that is passed from the client to the server contains the column name and the rowkey of the table cell to copy the value from. Another information that may be useful on the server is the clientId of the cell renderer component that you can get by calling getClientId() on the JavaScript component handle. A use case for passing the clientId is to explicitly set the focus back on the selected component, which is not required in the sample provided with this article.

```
/*
 * A global variable that holds the cell handler of the table cell that
 * we want to copy to other rows
 */
var globalFieldCopy = null;
var globalLastVisitedField = null;

/*
 * Function is called to open a context menu dialog next to the
 * selected table cell. It also saves the selected text field
 * component reference for later use when pasting the value to
 * selected table rows
 */
function openPopup(popupId){
  return function(evt){
  evt.cancel();
  //get the field reference to copy value from

  txtField = evt.getSource();
  //just temporarily remember the field that had focus
  //when the popup menu opened. This is to ensure that
  //fields are only copied when the Copy context menu
  //option is used

  globalLastVisitedField = txtField;

  //the context popup menu should be launched in the table next
  //to the selected table cell. For this we need to get the cell
  //handler component's clientId
  var clientId = txtField.getClientId();
  //search popup from page root
  var popup = AdfPage.PAGE.findComponentByAbsoluteId(popupId);
  //align popup so it renders after the textfield
  var hints = {align:"end_before", alignId:clientId};
  popup.show(hints);
 }
}

/*
 * As an alternative - and probably more convenient option to
 * copy and paste cell values to other rows - we provide a keyboard
 * option to use instead of the context menu.
 *
 * ctrl+shift+c copies the field to copy the value from.
 * ctrl+shift+v copies the saved value to all selected fields
 *
 * The serverListenerId argument is the name of the af:serverListener
 * component. The serverListenerOwnerComponentId is the ID of the UI
 * component that has the server listener defined as a child component.
 */
function copyFromKeyboard(serverListenerId,
serverListenerOwnerComponentId){
  return function(evt){

  //the keyboard event gives us the keycode and the modifier keys the
  // user pressed
  var keyPressed = evt.getKeyCode();
  var modifier = evt.getKeyModifiers();

  //copy if ctrl+shift key is pressed together with the c-key
  var shiftCtrlKeyPressed = AdfKeyStroke.SHIFT_MASK |
  AdfKeyStroke.CTRL_MASK;
  if (modifier == shiftCtrlKeyPressed){
    if(keyPressed == AdfKeyStroke.C_KEY){
      //copy the selected field to paste values from
      globalFieldCopy= evt.getSource();
      //cancel keyboard event
      evt.cancel();
     }
    //paste
    else if(keyPressed == AdfKeyStroke.V_KEY){
      if(globalFieldCopy == null){
        //no value copied
        alert("No value copied. Please copy value first:
                Use ctrl+shift+c");
        evt.cancel();
      }
      else{
        //handle paste
        copyValueToSelectedRows(serverListenerId,
        serverListenerOwnerComponentId);
        evt.cancel();
      }
    }
  }
 }
}
//Function referenced from the clientListener on the copy menu option
function copyFromMenu(evt){
  //copy the last visited field to the variable used to copy values
from. This
  //step is required to ensure the field is available when copy is
invoked from
  //popup
  if(globalLastVisitedField != null){
    globalFieldCopy = globalLastVisitedField;
   }
   else{
    alert("Problem: No field could be identified to be in focus");
   }
   //cancel keyboard event
    evt.cancel();
  }

//Function referenced from the clientListener on the copy menu option
 function pasteFromMenu(serverListenerId,
serverListenerOwnerComponentId){
   return function(evt){
     if(globalFieldCopy == null){
     //no value copied
     alert("No value copied. Please copy value first");
     evt.cancel();
     }
     else{
       //handle paste
       copyValueToSelectedRows(serverListenerId,
                               serverListenerOwnerComponentId);
       evt.cancel();
     }
   }
 }

/*
 * Function that queues a custom event to be handled by a
 * managed bean on the server
 * side. Note that having the copy and paste action handled
 * on the server is more
 * robust than doing the same on the client.
 */
function copyValueToSelectedRows(serverListenerId,
serverListenerOwnerComponentId) {
  var txtField = globalFieldCopy;
  //get the name of the column which row cell to read and write to
  var column = txtField.getProperty('colname');
  //get the index of the row to copy values from
  var rowKeyIndx = txtField.getProperty('rwKeyIndx');

  var submittedValue = txtField.getSubmittedValue();
  var serverListenerHolder = AdfPage.PAGE.findComponentByAbsoluteId
                             (serverListenerOwnerComponentId);
  AdfCustomEvent.queue(
   //reference the component that has the server listener
   //defined.
   serverListenerHolder,
   //specify server listener to invoke
   serverListenerId,
   // Send two parameters. The format of this message is
   //a JSON map, which on the server side Java code becomes
   //a java.util.Map object
   {column:column, fieldValue:submittedValue, rowKeyIndex:rowKeyIndx},
   // Make it "immediate" on the server
   true);

  //reset field value
   globalFieldCopy = null;
 }
```

Hint: A useful tool when programming JavaScript is the Firebug plugin in Firefox, which is a rock star when it comes to debugging. Even if you are deploying on IE for production, Firefox is a must have during development just for Firebug.

The use of af:serverListener and the managed bean calls

The server listener component allows developers to call server side Java from JavaScript on the client. In this example, the server listener calls the **copyValueToSelectedRows** method, passing the rowkey and the column name of the table cell to copy from. The method then reads the table cell value from the ADF Faces table to then iterate over the selected table rows to update the row attribute with the copied value. Optionally, and only when the business service is not ADF Business Components, you may call a merge or persist function to save the value changes in the business service.

The real work, the update of the table rows, is done in the private **updateSelectedTableRows** method. At the end of the method, a partial refresh is executed to show the updated table rows. Note that tables in ADF Faces are stamped so there is no way to just refresh the updated table rows. Interesting in the updateSelectedTableRows method is the use of DCDataRow, which may look strange for the ADF Business Components developers among the readers, because they are used to cast the ADF iterator rows to oracle.jbo.Row. DCDataRow is the implementation independent alternative and can be used with any DataControl, which is a benefit if you develop application code that is for reuse on other projects, which may not use ADF Business Components as a business service. Both, oracle,jbo.Row and DCDataRow extend RowImpl and expose the same set of common methods.

```
import java.util.Iterator;
import java.util.List;
import oracle.adf.model.BindingContext;
import oracle.adf.model.bean.DCDataRow;
import oracle.adf.model.binding.DCBindingContainer;
import oracle.adf.view.rich.component.rich.data.RichTable;
import oracle.adf.view.rich.context.AdfFacesContext;
import oracle.adf.view.rich.render.ClientEvent;
import oracle.binding.OperationBinding;
import oracle.jbo.uicli.binding.JUCtrlHierNodeBinding;
import org.apache.myfaces.trinidad.model.RowKeySet;

/**
 * Bean that handles the POJO model update
 */
public class EmployeesPageBean {

    private RichTable employeeTable1;

    public EmployeesPageBean() {
    }

   public void setEmployeeTable1(RichTable employeeTable1) {
       this.employeeTable1 = employeeTable1;
   }

   public RichTable getEmployeeTable1() {
       return employeeTable1;
   }

   /**
     * Method called from the serverListener on the page. The
     * values that are passed by the ClientEvent object is
     *customizable and in this sample include the row index and
     * the cell attribute name to copy and paste values from. You
     * could also pass the component clientId so that the managed
     * bean e.g. could set the focus back if needed
     * @param ce - Event object passed from the client to the server
     *   through the af:serverListener. The event objectcontains the
     *   message payload and a reference to the source invoking
     *   the server listener.
     */
   public void copyValueToSelectedRows(ClientEvent ce){
      String columnToUpdate = (String)
      ce.getParameters().get("column");
      RichTable table = this.getEmployeeTable1();
      //important: to avoid an exception in the RowChangeManager, you
      //need to copy the current row key and keep it in a local
      //variable
      Object oldKey = table.getRowKey();
      try {
        //get the row index of the table cell to copy from
        int rowKeyIndex =
          ((Double)ce.getParameters().get("rowKeyIndex")).intValue();
          //call a private method to update the selected rows
          updateSelectedTableRows(columnToUpdate, rowKeyIndex);
      }
      catch(Exception e){
        e.printStackTrace();
      }
      finally {
         //whatever happened, set the current row back to the copied
         //rowkey. This is needed for the RowManager to work in ADF
         //Faces tables
         table.setRowKey(oldKey);
      }
  }

  /**
     * Method that identifies the table row to copy the cell value from
     * to then copy the value to all selected table rows. Optionally,
     * at the end of ths method you then update the POJO model (which
     * is an action not required if you use ADF BC because the binding
     * layer is updated and within the next submit automatically
     * updates the ADF BC model. However, this sample uses a POJO model
     * and as such we perform the update
     * @param column           The attribute name to copy the values
     *                         from and paste them to
     * @param rowKeyIndex      the row index of the ADF binding row to
     *                         copy from. This is used
     *                         if you want to copy the values from the
     *                         ADF binding layer and not use the value
     *                         from the browser client. Its also the
     *                         default in this
     *                         sample.
     */
  private void updateSelectedTableRows(String column, int rowKeyIndex){

      RichTable table = this.getEmployeeTable1();
```

```
/*
 * 1. Get access to the copied cell data. This information
 *    can be accessed directly on the ADF Faces table or
 *    through the binding layer.
 */

//set current table row to copy source
table.setRowIndex(rowKeyIndex);
JUCtrlHierNodeBinding rowBinding =
       (JUCtrlHierNodeBinding) table.getRowData();

/* DCDataRow extends ViewRowImpl, which extends RowImpl that is
   used with ADF BC View Objects and Entity Objects. This means
   that DCDataRow is the class to use if your code needs to run
   with ADF BC and non-ADF BC business services
 */
DCDataRow rowToCopyFrom = (DCDataRow) rowBinding.getRow();
Object copyValue = rowToCopyFrom.getAttribute(column);

//* END OF OPTIONAl

/*
 * 2. Copy the data to the selected table rows. Here we have two
 *    options: paste the value to the ADF Faces table or access
      the ADF binding layer (iterator).
 */

RowKeySet selectedRowKeySet = table.getSelectedRowKeys();
Iterator selectedRowKeySetIter = selectedRowKeySet.iterator();

while (selectedRowKeySetIter.hasNext()){
    List key = (List) selectedRowKeySetIter.next();
    //make row current in table
    table.setRowKey(key);
    JUCtrlHierNodeBinding selectedRowBinding =
                      (JUCtrlHierNodeBinding) table.getRowData();
    DCDataRow rowToUpdate =
                         (DCDataRow) selectedRowBinding.getRow();

    //copy the value from one attribute to the current selected
    rowToUpdate.setAttribute(column, copyValue);

    /*
     * 3. So far we did update the ADF iterator, which is
       sufficient if you use ADF BC because of its active data
       model. Other business services expose methods to persist
       changes, which needs to be explicitly called.
     *
     *    Note that you can only persist rows that are complete
     *    and don't miss required values. In this sample, we
     *    update existing rows only so that
     *    it is safe to update.
     */
    BindingContext bctx = BindingContext.getCurrent();
    DCBindingContainer bindings =
       (DCBindingContainer) bctx.getCurrentBindingsEntry();
```

```
          //method to update the POJO model. Note that for
          //Pojo, EJB and WS models, you need to explicity calls
          //a method to merge (update) or persist (commit) objects
          OperationBinding mergeOperation =
              bindings.getOperationBinding("mergeEmployee");
          //Since we use POJOs we need to pass the "real" object,
          //which we get from the DataProvider of the row. The same
          //is required for Web Services, EJB/JPA etc.
          mergeOperation.getParamsMap().put(
                       "emp",rowToUpdate.getDataProvider());
          //execute the method for update
          mergeOperation.execute();
          //any errors ?
          if(mergeOperation.getErrors().size() > 0){
            //print first error message
            System.out.println("An Error occured when updating row: " +
                                 mergeOperation.getErrors().get(0));
          }
        }
      //PPR the table to display copied values
      AdfFacesContext adffctx = AdfFacesContext.getCurrentInstance();
      adffctx.addPartialTarget(table);
    }
}
```

Download

You can download the Oracle JDeveloper 11g (11.1.1.3, also known as PS1 R2) sample workspace **from ADF Code Corner**.

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

No configuration is required as the model is POJO based and all data is contained in the model classes. When the table is rendered, use the context menu or the keyboard combination ctrl+shift+c to copy a value and the context menu or ctrl+shift.p to paste the value.

RELATED DOCOMENTATION

⊠	af:resource tag - http://download.oracle.com/docs/cd/E15523_01/apirefs.1111/e12419/tagdoc/af_resource.html
⊠	AdfKeyStroke JS API - http://download.oracle.com/docs/cd/E15523_01/apirefs.1111/e12046/oracle/adf/view/js/base/AdfKeyStroke.html
⊠	af:serverListener tag - http://download.oracle.com/docs/cd/E15523_01/apirefs.1111/e12419/tagdoc/af_serverListener.html
⊠	DCDataRow - JavaDoc http://download.oracle.com/docs/cd/E15523_01/apirefs.1111/e10653/oracle/adf/model/bean/ DCDataRow.html
⊠	"Oracle Fusion Developer Guide", McGraw Hill – Oracle Press, Frank Nimphius, Lynn Munsinger
⊠	Using JavaScript in ADF Faces applications - http://download.oracle.com/docs/cd/E15523_01/web.1111/b31973/af_event.htm#DAFEBFEE