# ADF Code Corner

## How-to handle and respond to mouse double clicks in ADF Faces tables

**Abstract:**

There is no event associated with a mouse double click performed on an ADF Faces table. A frequent requirement though is to use a double click, e.g. to bring up an edit form for the current table row, or to perform other actions on the table row. Though the mouse double click is not natively handled in ADF Faces, using JavaScript allows you to implement this feature.

This article shows how to open an edit form in response to a double click performed on an ADF bound table.

twitter.com/adfcodecorner

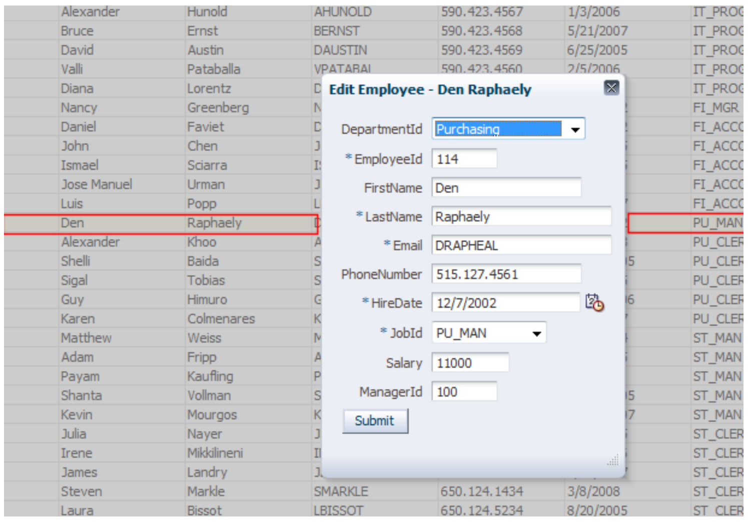
Author: Frank Nimphius, Oracle Corporation
twitter.com/fnimphiu
09-SEP-2010

## Introduction

A common user requirement is to perform a task on a tree, tree table or table row in response to a mouse double click. The ADF Faces table does not provide a native event listener to handle the mouse double click. However, this use case can be easily implemented with JavaScript.

### Use case: open edit form on table

The most frequent requirement for double click handling on a table, tree or tree table component is to open an edit form for the row clicked on.



| | | | | | |
|---|---|---|---|---|---|
| Alexander | Hunold | AHUNOLD | 590.423.4567 | 1/3/2006 | IT_PROG |
| Bruce | Ernst | BERNST | 590.423.4568 | 5/21/2007 | IT_PROG |
| David | Austin | DAUSTIN | 590.423.4569 | 6/25/2005 | IT_PROG |
| Valli | Pataballa | VPATABAL | 590.423.4560 | 2/5/2006 | IT_PROG |
| Diana | Lorentz | D | | | IT_PROG |
| Nancy | Greenberg | N | | | FI_MGR |
| Daniel | Faviet | D | | | FI_ACCO |
| John | Chen | J | | | FI_ACCO |
| Ismael | Sciarra | I | | | FI_ACCO |
| Jose Manuel | Urman | J | | | FI_ACCO |
| Luis | Popp | L | | | FI_ACCO |
| Den | Raphaely | D | | | PU_MAN |
| Alexander | Khoo | A | | | PU_CLER |
| Shelli | Baida | S | | 5 | PU_CLER |
| Sigal | Tobias | S | | | PU_CLER |
| Guy | Himuro | G | | 6 | PU_CLER |
| Karen | Colmenares | K | | | PU_CLER |
| Matthew | Weiss | M | | | ST_MAN |
| Adam | Fripp | A | | | ST_MAN |
| Payam | Kaufling | P | | | ST_MAN |
| Shanta | Vollman | S | | 5 | ST_MAN |
| Kevin | Mourgos | K | | 7 | ST_MAN |
| Julia | Nayer | J | | | ST_CLER |
| Irene | Mikkilineni | I | | | ST_CLER |
| James | Landry | J | | | ST_CLER |
| Steven | Markle | SMARKLE | 650.124.1434 | 3/8/2008 | ST_CLER |
| Laura | Bissot | LBISSOT | 650.124.5234 | 8/20/2005 | ST_CLER |

**Edit Employee - Den Raphaely**

| | |
|---|---|
| DepartmentId | Purchasing |
| * EmployeeId | 114 |
| FirstName | Den |
| * LastName | Raphaely |
| * Email | DRAPHEAL |
| PhoneNumber | 515.127.4561 |
| * HireDate | 12/7/2002 |
| * JobId | PU_MAN |
| Salary | 11000 |
| ManagerId | 100 |

**Figure 1: Open an edit dialog on table double click to edit the selected row**

The implementation uses a hybrid approach of JavaScript and server side Java. The reason for this is that more complex requires component and ADF binding access that is easier to do on the server than the client. In addition it shows that a pure JavaScript approach is not required and instead JavaScript is used for what JavaServer Faces or ADF Faces don't provide.

The edit dialog sample makes use of server side Java for opening and closing the dialog and for the cancel button in the popup. The latter accesses the ADF binding layer for the table.

```
<af:table  value="#{bindings.allEmployees.collectionModel}"
           selectionListener="
            #{bindings.allEmployees.collectionModel.makeCurrent}"
```

```
          rowSelection="single" id="t1">
  <af:column ...>
      ...
  </af:column>
  <af:column ...>
      ...
  </af:column>
  ...
  <af:clientListener method="handleTableDoubleClick"
                     type="dblClick"/>
  <af:serverListener type="TableDoubleClickEvent"
                     method="#{empbean.handleTableDoubleClick}"/>
</af:table>
```

The page source above defines the table with its *af:clientListener* and *af:serverListener* child tags. The *af:serverListener* is called from the JavaScript function that is invoked by the *af:clientListener*. The client listener responds to the mouse double click on the table.

Handy for this use case, the double click marks the clicked row as current, set by the selection listener referenced from the table "selectionListener" property. To show the selected row in an edit form, you only need to to drag the same View Object as a form into a popup. In the sample, the popup is added as a child of the *af:form* tag.

Note: For better readability of your applications page code, it is best to put the popup dialog next to the component launching it.

```
<af:popup id="p1" binding="#{allEmployeesBean.editEmployees}"
          contentDelivery="lazyUncached">
  <af:dialog id="d2" type="none" itle=" …" resize="on">
    <af:panelFormLayout id="pfl1">
      <af:selectOneChoice ...>
        ...
      </af:selectOneChoice>
      <af:inputText value= ... >
        ...
      </af:inputText>
        ...
    </af:panelFormLayout>
  </af:dialog>
</af:popup>
```

## The JavaScript

The JavaScript function gets the table component handle from the event object. The table is passed as a reference component in the custom server event, issued by the *af:serverListener* on the page. The double click event itself does not need to be handled on the server, which is why it is cancelled. If you want the popup to be opened so it aligns with a specific component on a page, you can use the payload, which is empty in this sample, to pass information to the server side method.

```
<af:resource type="javascript">
 function handleTableDoubleClick(evt){
   var table = evt.getSource();
   AdfCustomEvent.queue(table, "TableDoubleClickEvent",{}, true);
   evt.cancel();
  }
</af:resource>
```

## The managed bean code

The method that handles the server listener event opens the popup dialog. No popup arguments are defined, in which case the dialog is launched in the center of the page.

```
public void handleTableDoubleClick(ClientEvent ce){
  RichPopup popup = this.getEditEmployees();
  //no hints means that popup is launched in the
  //center of the page
  RichPopup.PopupHints ph = new RichPopup.PopupHints();
  popup.show(ph);
}
```

Note: To be able to open a popup from Java is a new features added in Oracle JDeveloper 11.1.1.3. The `PopupHints` classused to set the popup alignment hints is an inner class of the `RichPopup` class, which is not obvious.

The onSubmitPopup method hides the displayed popup dialog and refreshes the table component to show the user edits.

```
public void onSubmitPopup(ActionEvent actionEvent) {
   RichPopup popup = this.getEditEmployees();
   popup.hide();
   //refresh the table
   AdfFacesContext adfctx = AdfFacesContext.getCurrentInstance();
   adfctx.addPartialTarget(getEmployeesTable());
}
```

The "onCancel" action listener hides the popup dialog but also reset the user changes by calling "refresh" on the current selected row. Note that the table handle is sufficient to access the ADF binding and the current selected row of the iterator binding.

```
public void onCancel(ActionEvent actionEvent) {
  //undo changes
  RichTable  table = this.getEmployeesTable();
  CollectionModel model = (CollectionModel) table.getValue();
  JUCtrlHierBinding treeBinding =
                    (JUCtrlHierBinding) model.getWrappedData();
  DCIteratorBinding iterator =
                             treeBinding.getDCIteratorBinding();
  Row rw = iterator.getCurrentRow();
  rw.refresh(Row.REFRESH_UNDO_CHANGES);

  RichPopup popup = this.getEditEmployees();
  popup.hide();
}
```

## Sample Download

An Oracle JDeveloper 11.1.1.3 sample workspace can be downloaded from the ADF Code Corner website: [http://www.oracle.com/technetwork/developer-tools/adf/learnmore/index-101235.html](http://www.oracle.com/technetwork/developer-tools/adf/learnmore/index-101235.html)

**RELATED DOCOMENTATION**

| | |
|---|---|
| ⊠ | |
| ⊠ | |
| ⊠ | |