

ADF Code Corner

60. Table Drag and Drop Sample

ORACLE®
CODE CORNER



twitter.com/adfcodecorner

Abstract:

The Oracle ADF Faces JavaServer Faces UI component set provides additional framework functionality, which drag and drop is a feature of. Drag and drop can be designed declaratively for attributes, components and collections used as component models in table and trees structures.

In this blog article we describe drag and drop in a table, by example of row delete. Because, you can perform any action on the drag source and transferred object once you receive the drop event, the provided sample is a sufficient starting point for all kinds of operation.

Author:

Frank Nimphius, Oracle Corporation
twitter.com/fnimphiu
11-OCT-2010

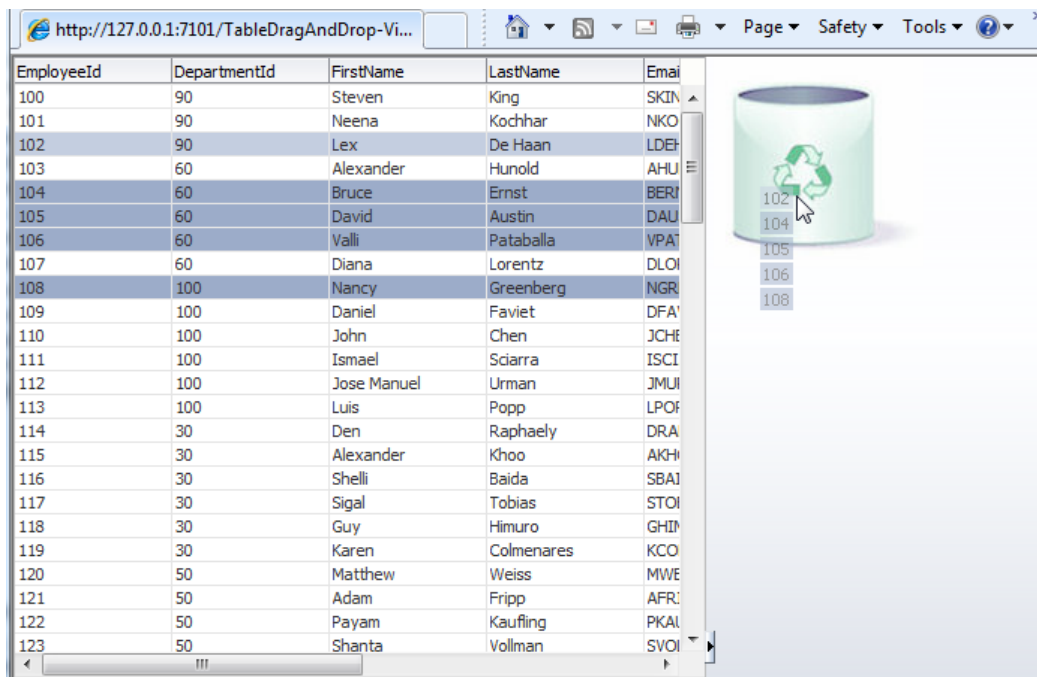
Oracle ADF Code Corner is a loose blog-style series of how-to documents that provide solutions to real world coding problems.

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.

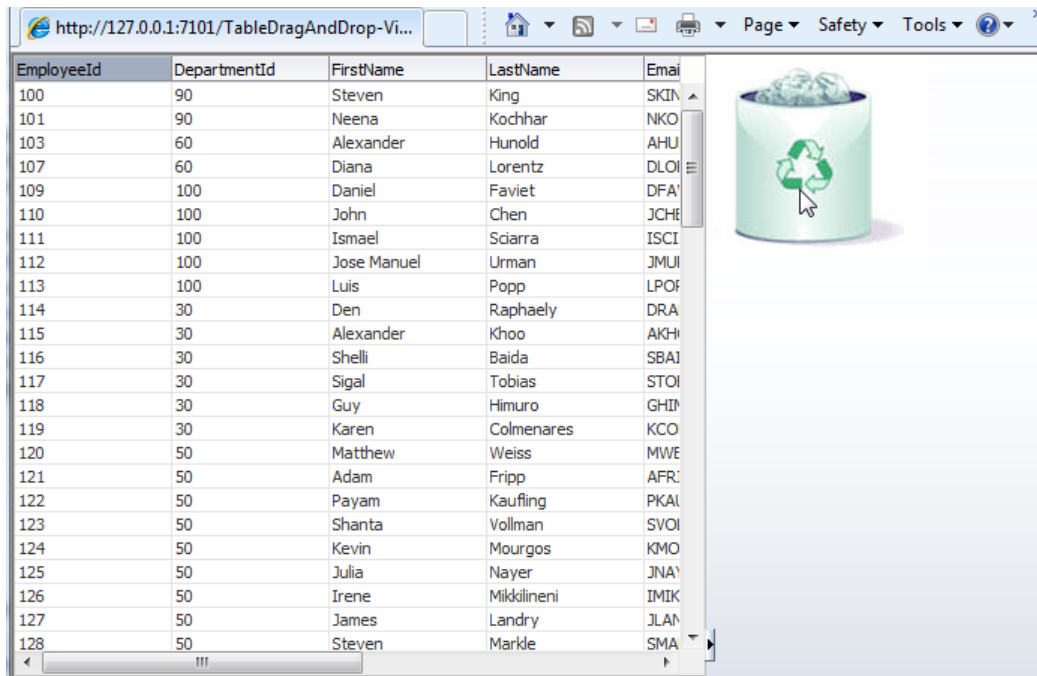
Please post questions or report problems related to the samples in this series on the OTN forum for Oracle JDeveloper: <http://forums.oracle.com/forums/forum.jspa?forumID=83>

Introduction

The drag and drop use case in this article is a user dragging one or many (using the ctrl key when selecting a row) rows from a table to a garbage bin image in a panel splitter. The garbage bin then acknowledges the delete with a short animation.



The little animation shown below is implemented using an `af:switcher` component and an `af:poll` component, which after 500 ms sends a request to the server to switch the image back to the "empty bin" to receive the next drag. Note that while the "full bin" image is shown, no new drop operation is possible. This demonstrates how developers can conditionally enable/disable drag and drop using the `af:switcher` component.



The screenshot shows a web browser window with the URL `http://127.0.0.1:7101/TableDragAndDrop-Vi...`. The browser displays a table with the following data:

EmployeeId	DepartmentId	FirstName	LastName	Email
100	90	Steven	King	SKIN
101	90	Neena	Kochhar	NKO
103	60	Alexander	Hunold	AHU
107	60	Diana	Lorentz	DLOI
109	100	Daniel	Faviet	DFA
110	100	John	Chen	JCH
111	100	Ismael	Sciarra	ISCI
112	100	Jose Manuel	Urman	JMUJ
113	100	Luis	Popp	LPOF
114	30	Den	Raphaely	DRA
115	30	Alexander	Khoo	AKH
116	30	Shelli	Baida	SBAJ
117	30	Sigal	Tobias	STOI
118	30	Guy	Himuro	GHIM
119	30	Karen	Colmenares	KCO
120	50	Matthew	Weiss	MWE
121	50	Adam	Fripp	AFR
122	50	Payam	Kaufling	PKAL
123	50	Shanta	Vollman	SVOI
124	50	Kevin	Mourgos	KMO
125	50	Julia	Nayer	JNAY
126	50	Irene	Mikkilineni	IMIK
127	50	James	Landry	JLAN
128	50	Steven	Markle	SMA

To the right of the table is a trash bin icon with a recycling symbol, indicating a drop target for the table's data.

The two techniques to learn from this article are

- Using drag and drop on a table
- Using `af:switcher` to conditionally enable/disable drag and drop

Implementing the drag operation

The drag operation is configured on the table by dragging the `af:dragSource` component as shown below

```
<af:table ... rowSelection="multiple">
  <af:dragSource actions="MOVE" discriminant="rowmove"
    defaultAction="MOVE"/>
  ...
</af:table>
```

The `actions` attribute and the `discriminant` attribute are used like name spaces and allow developers to ensure drag and drop is performed between components that are meant to work together. You can have multiple instance of drag and drop added on a single page without risking a conflict due to users dragging the wrong object to a specific drop spot. The `actions` attribute takes the following arguments: COPY, MOVE, LINK. These actions define the capability of a drag source, information developers can work with in Java and JavaScript. In our use case any of these actions would do.

Implementing the drop operation

The drop area is defined by the `af:dropTarget` tag. In the example, this tag is added as a child component to the `af:image` tag rendering the garbage bin

```
<af:image source="/images/gbin1.jpg" id="i1">
  <af:dropTarget dropListener="#{DropHandlerBean.dropHandler}"
  ...
</af:image>
```

```
        actions="MOVE">
      <af:dataFlavor flavorClass="org.apache.myfaces.trinidad.model.RowKeySet"
        discriminant="rowmove"/>
    </af:dropTarget>
</af:image>
```

The drop target uses three attributes to define the acceptable drag operation: "action", "flavorClass" and "discriminant".

The *action* attribute of the drop target must match the drag action. Both, the drag and the drop tag may define multiple actions, e.g. COPY MOVE, in which case only one of the listed actions need to match for the drop operation to be accepted. The *discriminant* attribute is defined for the data flavor sub tag of the dropTag. The discriminant must match the discriminant used on the drag source. The flavorClass provides a hint about the payload sent from the client to the server, which in the case of tables and tree always is the RowKeySet. The dropTarget tag has a *dropListener* attribute configured, which points to a managed bean method to handle the drop event. The code of the managed bean method is shown and explained in the next section.

Handling the drop event

The drop handler method receives an argument of DropEvent, which contains information about the drag source (the table) and the drop target (the image), as well as the transferred object (the RowKeySet).

```
public DnDAction dropHandler(DropEvent dropEvent) {
    //get the drag source
    RichTable table = (RichTable) dropEvent.getDragComponent();
    //determine the rows that are dragged over
    Transferable t = dropEvent.getTransferable();
    //when looking for data, use the same discriminator as defined
    //on the drag source
    DataFlavor<RowKeySet> df =
        DataFlavor.getDataFlavor(RowKeySet.class, "rowmove");
    RowKeySet rks = t.getData(df);
    Iterator iter = rks.iterator();
    while (iter.hasNext()) {
        //get next selected row key
        List key = (List)iter.next();
        //make row current so we can access it
        table.setRowKey(key);
        //the table model represents its row by the ADF binding class,
        //which is JUCtrlHierNodeBinding
        JUCtrlHierNodeBinding rowBinding =
            (JUCtrlHierNodeBinding) table.getRowData();
        Row row = (Row) rowBinding.getRow();
        //delete row
        row.remove();
        //activate animation
    }
}
```

```
Map viewScope = AdfFacesContext.getCurrentInstance().getViewScope();
//show different icon for bin collector. The new icon
//does not take drop payloads and is shown for 1 second
viewScope.put("binState", "fullbin");
//refresh garbage bin parent (panelGroupLayout)
refreshComponentContainer("pgl1");
AdfFacesContext.getCurrentInstance().addPartialTarget(table);
//table source is manually refreshed, so we can return NONE.
//Returning MOVE instead would implicitly refresh the drop source
return DnDAction.NONE;
}

//helper method to find a component and refresh it using PPR
private void refreshComponentContainer(String id){
    String comp = id;
    //see ADF Code Corner sample #58 to read about the code
    //used in the search below
    FacesContext fctx = FacesContext.getCurrentInstance();
    UIViewRoot root = fctx.getViewRoot();
    root.invokeOnComponent(fctx, comp,
        new ContextCallback(){
            public void invokeContextCallback(FacesContext facesContext,
                UIComponent uiComponent) {
                //refresh parent container
                AdfFacesContext.getCurrentInstance().addPartialTarget(uiComponent);
            }
        });
}
```

What the code above does is to access the RowKeySet containing information about the dragged table rows, iterating over the row keys to access the row they represent and calling remove to delete them. It then refreshes the table and the panelGroupLayout component that holds the garbage bin image. The drop handle sets the af:switcher facet value for the "full bin" image to show in acknowledgement of the delete operation.

Implementing the animation

To indicate a successful drop operation, the garbage bin image changes from



to



And back. This animation is realized using an `adf:switcher` component that reads from a parameter in `viewScope` upon refresh.

```
<af:panelGroupLayout layout="scroll" id="pgl1">
  <adf:switcher id="s1" defaultFacet="emptybin"
    facetName="#{viewScope.binState}">
    <f:facet name="emptybin">
      <af:panelGroupLayout id="pgl2" layout="scroll">
        <af:image source="/images/gbin1.jpg" id="i1">
          <af:dropTarget dropListener="#{DropHandlerBean.dropHandler}"
            actions="MOVE">
            <af:dataFlavor flavorClass="org.apache.myfaces.trinidad.model.RowKeySet"
              discriminant="rowmove"/>
          </af:dropTarget>
        </af:image>
      </af:panelGroupLayout>
    </f:facet>
    <f:facet name="fullbin">
      <af:panelGroupLayout id="pgl3" layout="scroll">
        <af:image source="/images/gbin2.jpg" id="i2"/>
        <af:poll id="p1" interval="500"
          pollListener="#{DropHandlerBean.onPoll}"
          clientComponent="true"/>
      </af:panelGroupLayout>
    </f:facet>
  </adf:switcher>
</af:panelGroupLayout>
```

The `adf:switcher` component has two facets. The first facet contains the empty garbage bin image and the drop listener to handle the drag and drop operation. The second facet shows the "full" garbage bin image and has an `af:poll` component that fires after 500 ms to switch the image back to the empty garbage bin image. The poll listener code is shown below

```
public void onPoll(PollEvent pollEvent) {
    Map viewScope = AdfFacesContext.getCurrentInstance().getViewScope();
    //show different icon for bin collector. The new icon
    //does not take drop payloads and is shown for 1 second
    viewScope.put("binState", "emptybin");
    //refresh garbage bin parent (panelGroupLayout)
    refreshComponentContainer("pgl1");
}
```

Switching back to the other facet also stops the poll so that it fires only one time.

Download

You can download the Oracle JDeveloper 11.1.1.3 (R1 PS1) workspace from the ADF Code Corner website at <http://www.oracle.com/technetwork/developer-tools/adf/learnmore/index-101235.html>.

Configure the model project to access a database with the HR schema installed and enabled. Then run the sample and drag table rows from the table to the garbage bin. Note that though rows are deleted, the sample doesn't commit the change to preserve your data. If you want to commit the change, simply drag and drop the "Commit" operation as a button from the Data Control panel to the page and press it after the drop.

RELATED DOCUMENTATION

<input type="checkbox"/>	Drag and Drop: Product Documentation http://download.oracle.com/docs/cd/E15523_01/web.1111/b31973/af_dnd.htm#CIHCHGIF
<input type="checkbox"/>	Chapter 14: Oracle Fusion Developer Guide – McGraw Hill Oracle Press, Frank Nimphius, Lynn Munsinger http://www.mhprofessional.com/product.php?cat=112&isbn=0071622543
<input type="checkbox"/>	