ADF Code Corner

103. How-to edit an ADF form with data dragged from an ADF Faces table

twitter.com/adfcodecorner

Abstract:

Drag and drop appears to be a feature area in ADF Faces that is not widely used by Oracle ADF application developers. One reason for this might be the lack of documented examples for specific use cases.

So here's another one: In this article I explain how to drag and drop a row from an ADF Faces table on top of an input form to copy data into the form.

Author: Frank Nimphius, Oracle Corporation
twitter.com/fnimphiu
22-AUG-2012

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

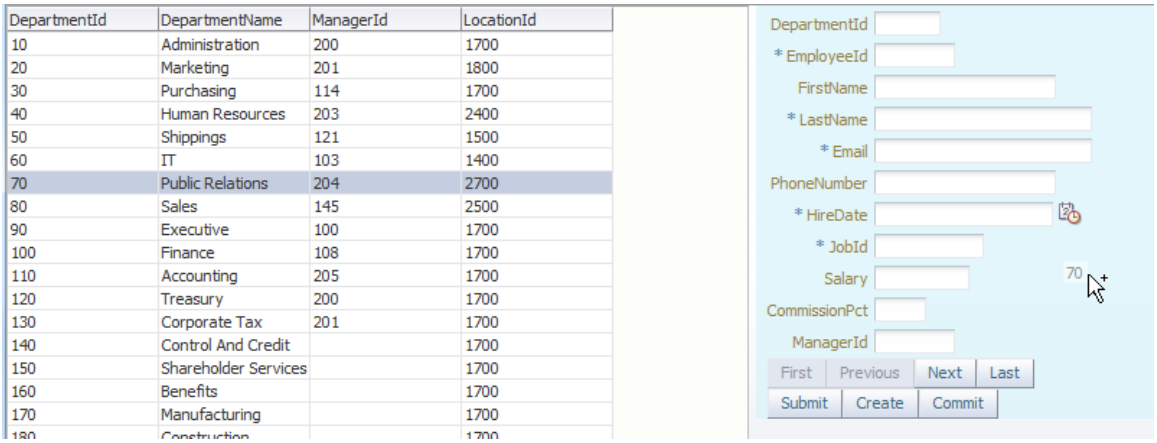
Disclaimer: All samples are provided as is with no guarantee for future upgrades or error correction. No support can be given through Oracle customer support.

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

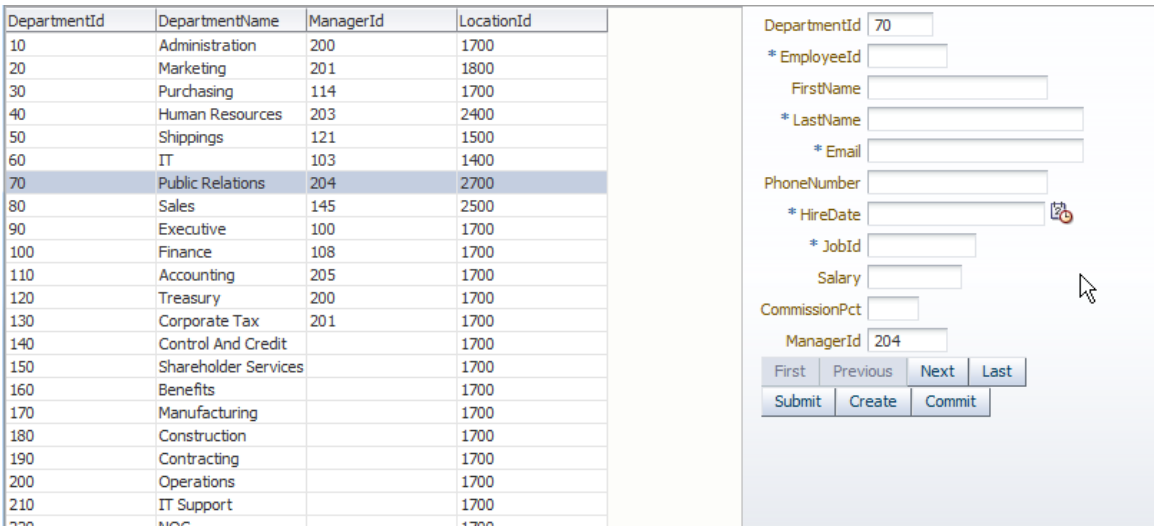
In a previous article, "How-to drag-and-drop data from an af:table to af:tree", I explained how to update a tree with data dragged from an ADF Faces table. In this sample, I am using the same code base but perform the drop operation onto an iput for to update or create new employees.

As in the screenshot below, the drag operation is originated from an ADF Faces table that has single row selection enabled. The drop area is a **panelFormLayout** component that then invokes the droplistener to process the data exchange. The sample can be easily changed to support drag and drop to other layout containers holding the form input fields.



The screen shot below shows the result of the table row being dropped onto an empty form. The employee DepartmentId and ManagerId fields are updated by the drop operation (in practice it is up to you how many attributes exposed in form fields you update using drag and drop).

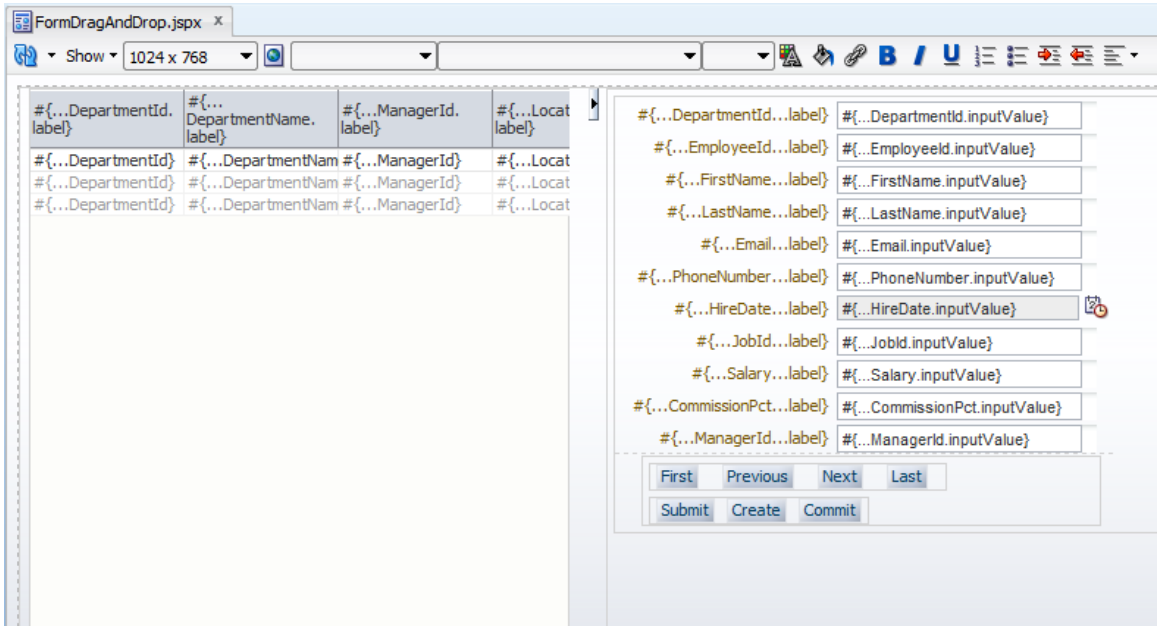
A problem that showed when updating the form for a new employee record is that the drop operation refreshes the drop target – the panelFielLayout component – which then refreshes its child components. This lead to many required field warnings that the sample code posted in this article needed to suppress using JavaScript.



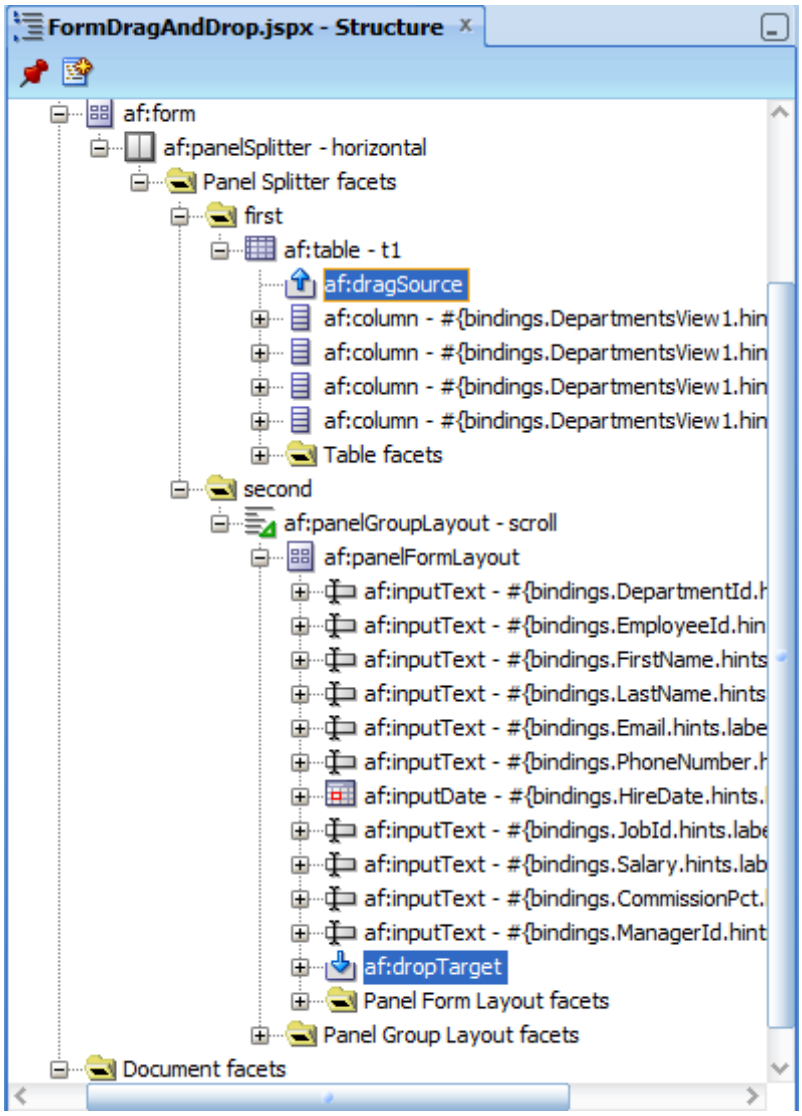
DepartmentId	DepartmentName	ManagerId	LocationId
10	Administration	200	1700
20	Marketing	201	1800
30	Purchasing	114	1700
40	Human Resources	203	2400
50	Shippings	121	1500
60	IT	103	1400
70	Public Relations	204	2700
80	Sales	145	2500
90	Executive	100	1700
100	Finance	108	1700
110	Accounting	205	1700
120	Treasury	200	1700
130	Corporate Tax	201	1700
140	Control And Credit		1700
150	Shareholder Services		1700
160	Benefits		1700
170	Manufacturing		1700
180	Construction		1700
190	Contracting		1700
200	Operations		1700
210	IT Support		1700

Building the Sample

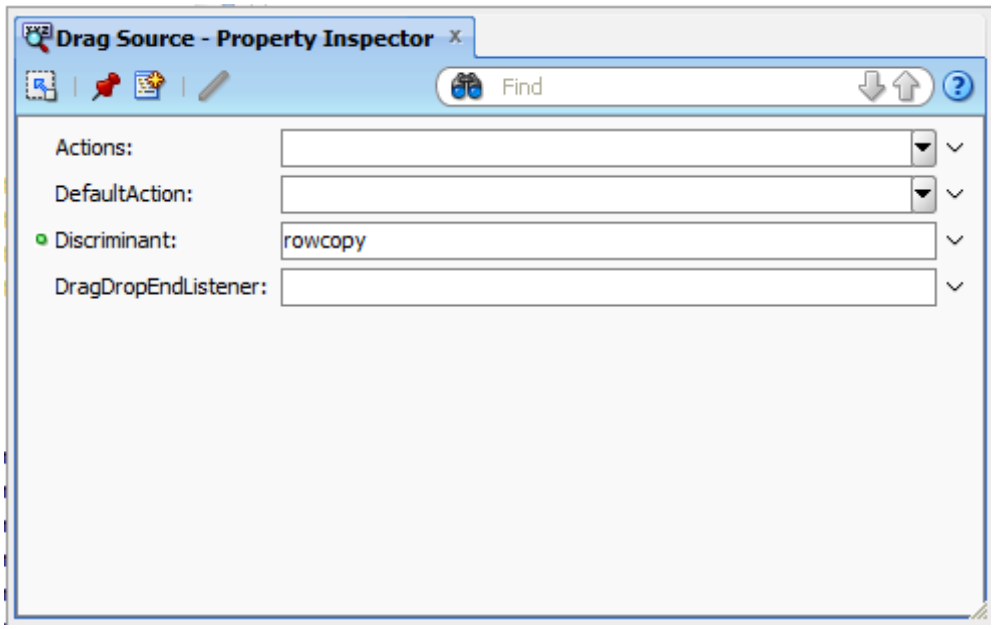
The sample itself is built based on the Oracle HR schema using ADF Business Components as the business service. In an ADF Faces page, a **panelSplitter component** is used to hold the drag source (table) and the drop target (input form).



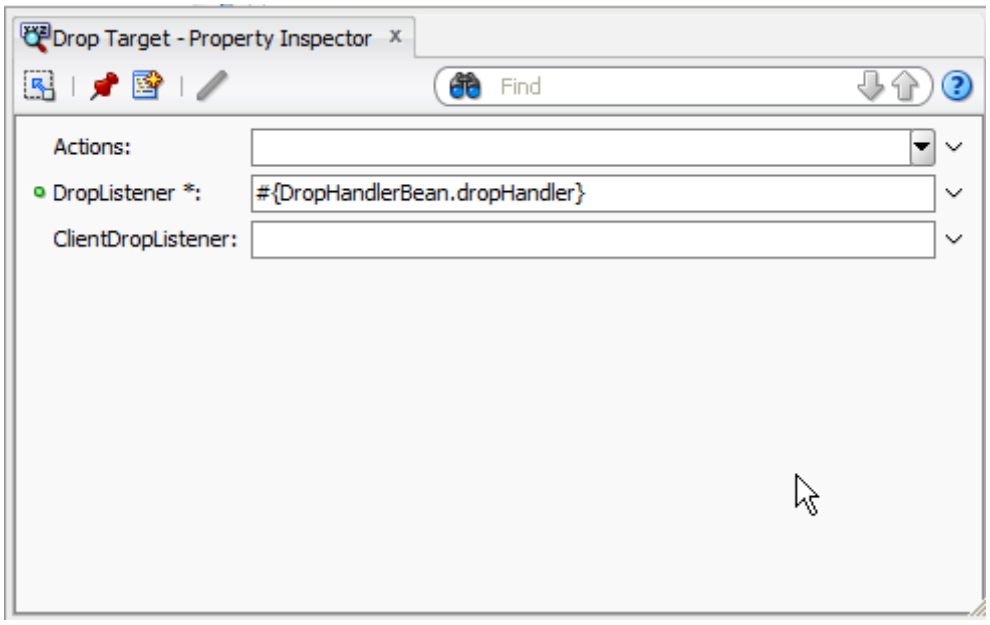
As shown in the next image, to enable drag and drop in ADF Faces, you add the **af:dragSource** component to the table and the **af:dropTarget** component to the panelFormLayout component. The two behavior tags implement all the JavaScript required at runtime to perform the drag and drop operation.



The configuration of the af:dragSource is shown below: The **Actions** property when left empty defaults to the **Copy** action (Move and Link would be the other choices). The **Discriminant** property defines a key string that ensures the drag operation to be answered only by drop targets that know how to deal with it. Using a discriminator allows you to have multiple drag-and-drop tags on a page without the application user to accidentally drop a collection to the wrong target.



The drop target tag also requires less configuration and defaults to **COPY** as the default action. Explicitly setting the Actions property value makes sense for components that support other actions like **MOVE** and **Link** too.



The **DropListener** property points to a managed bean to receive the drop event. The managed bean method then also receives access to all information, including the transferred data object, the drag source and the drop target components, required to handle the event. The **ClientDropListener** property is only useful if you want to suppress the drop event based on a client side condition evaluated in JavaScript.

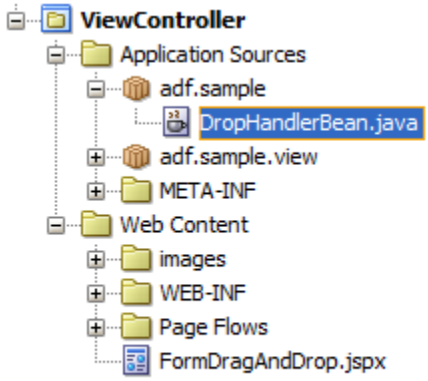
The **DropTarget** configuration that is not shown in the screenshot is the definition of the data flavor, the type of data transferred with the drag-and-drop operation.

In the sample the whole **DropTarget** tag definition added as a child to the **PanelFormLayout** looks as shown below

```
<af:dropTarget dropListener="#{DropHandlerBean.dropHandler}">
    <af:dataFlavor flavorClass="org.apache.myfaces.trinidad.model.RowKeySet"
                   discriminant="rowcopy"/>
</af:dropTarget>
```

The data type to read from the transfer object is the RowKeySet of the table row(s) dragged from the ADF Faces table. The **dataFlavor** tag also defines a discriminator that matches the key string defined on the **DragSource**.

On the view layer, these few configurations is all you need to do to get drag-and-drop working in ADF Faces. The rest is all handled in Java within a managed bean (DropHandlerBean in this sample). Drag and drop really is easy to use and I am surprised why the feature is not widely used.



DropHandlerBean code

The best way to explain code is to show it and have comments speaking for it. The use case in the sample is that upon drag and drop (a single row can be dragged at a time) of a table row, the department Id and the manager Id are copied as a value to the current employee record (which can be a new record).

As a rule employees should not report to themselves but someone else.

```
package adf.sample;

import java.util.Iterator;
import java.util.List;
import javax.faces.component.UIComponent;
import javax.faces.context.FacesContext;
import oracle.adf.model.BindingContext;
import oracle.adf.model.binding.DCIteratorBinding;
import oracle.adf.view.rich.component.rich.data.RichTable;
import oracle.adf.view.rich.component.rich.input.RichInputText;
import oracle.adf.view.rich.component.rich.layout.RichPanelFormLayout;
```

```
import oracle.adf.view.rich.datatransfer.DataFlavor;
import oracle.adf.view.rich.datatransfer.Transferable;
import oracle.adf.view.rich.dnd.DnDAction;
import oracle.adf.view.rich.event.DropEvent;
import oracle.binding.BindingContainer;
import oracle.jbo.Row;
import oracle.jbo.uicli.binding.JUCtrlHierNodeBinding;
import org.apache.myfaces.trinidad.model.RowKeySet;
import org.apache.myfaces.trinidad.render.ExtendedRenderKitService;
import org.apache.myfaces.trinidad.util.Service;


/**
 * Drag and drop handler that handles the table row drop onto
 * an input form
 * @author Frank Nimphius (8/2012)
 */
public class DropHandlerBean {
  public DropHandlerBean() {
  }
  //method referenced from the af:dropTarget
  public DnDAction dropHandler(DropEvent dropEvent) {
    //access the drag source, the table to read the rowKey
    //representing the row
    RichTable table = (RichTable)dropEvent.getDragComponent();
    Transferable t = dropEvent.getTransferable();
    DataFlavor<RowKeySet> df =
             DataFlavor.getDataFlavor(RowKeySet.class,"rowcopy");
    RowKeySet rks = t.getData(df);
    Iterator iter = rks.iterator();
    //sample is set up for single row drag and drop
    if (iter.hasNext()) {
      //get next selected row key
      List key = (List)iter.next();
      table.setRowKey(key);

      //get handle to drop component
      RichPanelFormLayout panelFormLayout =
             (RichPanelFormLayout) dropEvent.getDropComponent();
      //JUCtrlHierNodeBinding is the object in ADF that represents
      //a row in a table or a node in a tree. The object wraps the
      ///actual table row object, which in the case of ADF BC is
```

```
 //oracle.jbo.Row
 JUCtrlHierNodeBinding rowBinding =
      (JUCtrlHierNodeBinding)table.getRowData();

 //the row that is actually dragged from the ADF Faces table
 Row departmentDropRow = rowBinding.getRow();

 //update current row in form if the table row is not null
if(departmentDropRow != null){
  //to evaluate and update the manager Id field, we need to create
  //and store a handle to it. The bean is in request (or backing
  //bean scop, so no overhead)
  UIComponent managerId = null;

  //search all panelFormLayout component children for the
  //DepartmentId field "it8" and ManagerId field "it2". This
  //is where you can extend the sample in your implementation to
  //update more than 2 fields
  for(UIComponent uiComp : panelFormLayout.getChildren()){
    if(uiComp.getId()=="it8"){
       ((RichInputText)uiComp).resetValue();
       ((RichInputText)uiComp).setValue(
               departmentDropRow.getAttribute("DepartmentId"));

    }
    else if(uiComp.getId()=="it2"){
       //save component handle for later (see below)
       managerId = uiComp;

    }
  }

   //access current binding container
   BindingContext bctx = BindingContext.getCurrent();
   BindingContainer bindings = bctx.getCurrentBindingsEntry();
   //get current employee row
   DCIteratorBinding employeeIterator =
       (DCIteratorBinding) bindings.get("EmployeesView1Iterator");
   Row currentEmployee = employeeIterator.getCurrentRow();
  //managers should not manage themselves
  oracle.jbo.domain.Number employeeId = (oracle.jbo.domain.Number)
                       currentEmployee.getAttribute("EmployeeId");

   oracle.jbo.domain.Number deptManagerId =
                    (oracle.jbo.domain.Number)departmentDropRow.
                     getAttribute("ManagerId");
```

```
        if(employeeId == null ||
          (employeeId.intValue() != deptManagerId.intValue())){
           ((RichInputText)managerId).resetValue();
           ((RichInputText)managerId).setValue(
                 departmentDropRow.getAttribute("ManagerId"));
            }
          }
           //at the end of the drag and drop operation, the drop target,
           //the panelFormLayout in this case, is refreshed. If the form
           //you edit with drag and drop contains required fields, then
           //these would be flagged as an error.
           //To avoid this in this sample, JavaScript is used to clear
           //all error messages on the client that occur in response to
           //the drag and drop operation
            FacesContext fctx = FacesContext.getCurrentInstance();
            ExtendedRenderKitService erks =
                  Service.getRenderKitService(fctx,
                                        ExtendedRenderKitService.class);
            erks.addScript(fctx,"AdfPage.PAGE.clearAllMessages();");
            //success ! So acknowledge the COPY for the drop target
            //to refresh
            return DnDAction.COPY;
          }
           //no drag and drop happened
         return DnDAction.NONE;
        }
}
```

The code lines highlighted in bold and red are those you want to pay attention to. This is the trick that allows drag and drop to happen of form fields that – after the drop operation – have required fields with no value entry. This is to avoid required field values to show.

Summary and Download

This article explained how to implement drag and drop between an ADF Faces table and an input form. The sample code is for JDeveloper 11g R1 (11.1.1.6) but also works with newer releases of Oracle JDeveloper.

To run the sample, configure the database connection of the application to access the HR schema in a local Oracle database installation.

The workspace for this sample can be downloaded as sample #103 from the ADF Code Corner website

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

RELATED DOCOMENTATION

☒	"How-to drag-and-drop data from an af:table to af:tree" http://www.oracle.com/technetwork/developer-tools/adf/learnmore/101-drag-drop-table-tree-1661895.pdf
☒	
☒	