

ADF Code Corner

101. How-to drag-and-drop data from an af:table to af:tree

ORACLE
CODE CORNER



twitter.com/adfcodecorner

Abstract:

Drag and drop is a feature of the ADF Faces framework. All developers have to do to make this work is to add a drag source tag to the component that is the origin of a drag operation and a drop tag to the component that is the destination of this operation.

This article provides a sample and explanation of how to develop drag and drop between an ADF Faces table and an ADF Faces tree component.

Author:

Frank Nimphius, Oracle Corporation
twitter.com/fnimphiu
13-JUN-2012

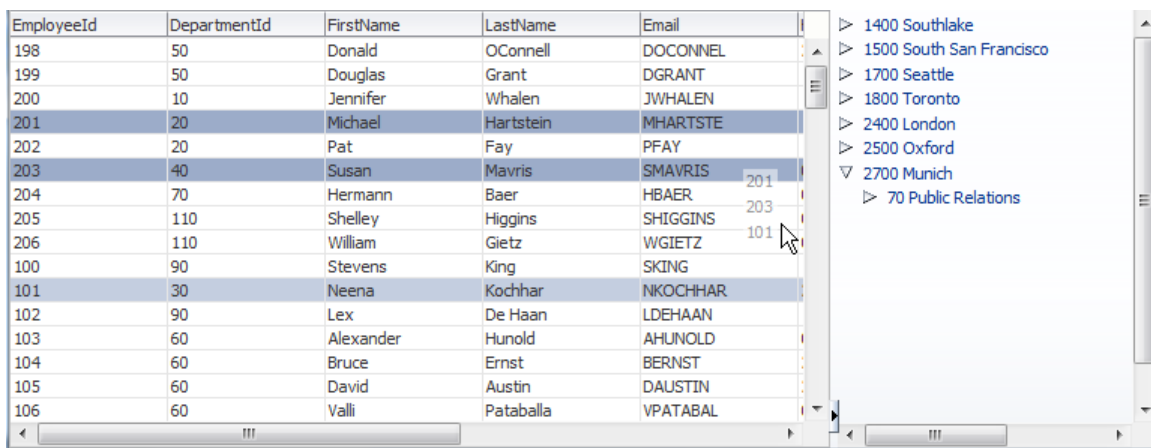
Oracle ADF Code Corner is a loose blog-style series of how-to documents that provide solutions to real world coding problems.

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

The Oracle JDeveloper 11g R1 (11.1.1.6) sample application you can download from a link at the end of this article shows a table of employees to the left and a tree component to the right.



The screenshot shows a JDeveloper application with two components: a table on the left and a tree component on the right. The table displays employee data with columns for EmployeeId, DepartmentId, FirstName, LastName, and Email. The tree component shows a hierarchy of departments, including 1400 Southlake, 1500 South San Francisco, 1700 Seattle, 1800 Toronto, 2400 London, 2500 Oxford, 2700 Munich, and 70 Public Relations. A mouse cursor is hovering over the row for employee 203 (Susan Mavris) in the table.

EmployeeId	DepartmentId	FirstName	LastName	Email
198	50	Donald	OConnell	DOCONNEL
199	50	Douglas	Grant	DGRANT
200	10	Jennifer	Whalen	JWHALEN
201	20	Michael	Hartstein	MHARTSTE
202	20	Pat	Fay	PFAY
203	40	Susan	Mavris	SMAVRIS
204	70	Hermann	Baer	HBAER
205	110	Shelley	Higgins	SHIGGINS
206	110	William	Gietz	WGIEZT
100	90	Stevens	King	SKING
101	30	Neena	Kochhar	NKOCHHAR
102	90	Lex	De Haan	LDEHAAN
103	60	Alexander	Hunold	AHUNOLD
104	60	Bruce	Ernst	BERNST
105	60	David	Austin	DAUSTIN
106	60	Valli	Pataballa	VPATABAL

You can drag one or multiple rows from the table and drop it onto the tree component. If the drop is to a node that is not a department, an information alert is shown



If the drop is on a department node then the employee accounts within the drop are updated with the department ID of the drop target.

EmployeeId	DepartmentId	FirstName	LastName	Email
198	50	Donald	OConnell	DOCONNEL
199	50	Douglas	Grant	DGRANT
200	10	Jennifer	Whalen	JWHALEN
201	20	Michael	Hartstein	MHARTSTE
202	20	Pat	Fay	PFAY
203	40	Susan	Mavris	SMAVRIS
204	70	Hermann	Baer	HBAER
205	110	Shelley	Higgins	SHIGGINS
206	110	William	Gietz	WGIETZ
100	90	Stevens	King	SKING
101	30	Neena	Kochhar	NKOCHHAR
102	90	Lex	De Haan	LDEHAAN
103	60	Alexander	Hunold	AHUNOLD
104	60	Bruce	Ernst	BERNST
105	60	David	Austin	DAUSTIN
106	60	Valli	Pataballa	VPATABAL

The tree is refreshed and the department node with the new employee nodes expanded.

EmployeeId	DepartmentId	FirstName	LastName	Email
198	50	Donald	OConnell	DOCONNEL
199	50	Douglas	Grant	DGRANT
200	10	Jennifer	Whalen	JWHALEN
201	70	Michael	Hartstein	MHARTSTE
202	20	Pat	Fay	PFAY
203	70	Susan	Mavris	SMAVRIS
204	70	Hermann	Baer	HBAER
205	110	Shelley	Higgins	SHIGGINS
206	110	William	Gietz	WGIETZ
100	90	Stevens	King	SKING
101	70	Neena	Kochhar	NKOCHHAR
102	90	Lex	De Haan	LDEHAAN
103	60	Alexander	Hunold	AHUNOLD
104	60	Bruce	Ernst	BERNST
105	60	David	Austin	DAUSTIN
106	60	Valli	Pataballa	VPATABAL

Note that the sample does not commit the data changes to preserve the state of your HR schema data.

The sample code has a couple of useful code examples, like

- Handling a drop event for a data collection
- Determine the node type of the drop
- Avoiding conflicts due to programmatic changes of the tree row state

- Expanding specific tree node path

Implementing drag and drop

The implementation of drag and drop between an ADF Faces table and a tree is not complicated at all, as you can tell from the page source shown below:

```
<af:table value="#{bindings.EmployeesView1.collectionModel}"
          var="row" rows="#{bindings.EmployeesView1.rangeSize}" ...">
  <af:dragSource actions="MOVE" discriminant="rowmove"
                defaultAction="MOVE"/>
  <af:column ...>
    ...
  </af:column>
```

The **af:dragSource** tag is an immediate child of the **af:table** component. The **discriminant** property allows you to associate a specific drag event with this drop target so that users cannot make mistakes in pages that support drag and drop on multiple components. The action MOVE provides an additional discriminator in that the drag only occurs on drop targets that also support the MOVE action. Other action types are COPY and LINK.

The drop component in this sample is the ADF Faces tree component. The configuration is shown below

```
<af:tree value="#{bindings.LocationsView11.treeModel}" var="node"
         selectionListener="#{bindings.LocationsView11.treeModel.makeCurrent}"
         rowSelection="single" id="t2">
  <f:facet name="nodeStamp">
    <af:outputText value="#{node}" id="ot11"/>
  </f:facet>
  <af:collectionDropTarget dropListener="#{DropHandlerBean.dropHandler}"
                          actions="MOVE" modelName="rowmove"/>
</af:tree>
```

As mentioned, the implementation of drag and drop in ADF Faces is not difficult. The drop target is defined by the **af:collectionDropTarget dropListener**, which is an immediate child of the **af:tree** component. The **modelName** property matches the **discriminant** property value of the drag source. If the action wasn't MOVE but COPY or LINK then no drop would be allowed. Note that you can define multiple actions, for example actions="COPY MOVE"

The drop handler code

If the implementation of drag and drop on the page is so simple, then there must be power within the managed bean method that handles the drop event. The beauty of drag and drop in ADF Faces is that it

takes the burden of cross-browser drag-and-drop JavaScript development from the application developer and instead delegates the event handling to Java in a managed bean.

```
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import javax.faces.component.ContextCallback;
import javax.faces.component.UIComponent;
import javax.faces.component.UIViewRoot;
import javax.faces.context.FacesContext;
import oracle.adf.model.bean.DCDataRow;
import oracle.adf.view.rich.component.rich.RichPopup;
import oracle.adf.view.rich.component.rich.data.RichTable;
import oracle.adf.view.rich.component.rich.data.RichTree;
import oracle.adf.view.rich.context.AdfFacesContext;
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.jbo.Row;
import oracle.jbo.uicli.binding.JUCtrlHierBinding;
import oracle.jbo.uicli.binding.JUCtrlHierNodeBinding;
import org.apache.myfaces.trinidad.event.PollEvent;
import org.apache.myfaces.trinidad.event.SelectionEvent;
import org.apache.myfaces.trinidad.model.CollectionModel;
import org.apache.myfaces.trinidad.model.RowKeySet;
import org.apache.myfaces.trinidad.model.RowKeySetImpl;

public class DropHandlerBean {

    //Variable holding the component looked up on a page. See
    // "findComponentOnPage" method in this class that allows
    // you to get a handle to any component in a view by the
    // component ID using an optimized search explained in ADF
    // Code Corner sample #58
    // http://www.oracle.com/technetwork/developer-tools/adf/downloads/58-
    // optimizedadffacescomponentsearch-175858.pdf

    private UIComponent lookupComponent = null;

    public DropHandlerBean() {}

    public DnDAction dropHandler(DropEvent dropEvent) {
        //get handle to drag component
```

```
RichTable table = (RichTable) dropEvent.getDragComponent();
//get handle to drop component
RichTree tree = (RichTree) dropEvent.getDropComponent();
//memorize current row key to restore the original selection state
//of the tree to avoid issues with context menus or similar. Note
//that this step is important and should never be omitted when you
//programmatically change the row key for trees, tables and tree
//tables
Object currentRowKey = tree.getRowKey();

//the drop site tells us about the node that received the drop
//event. In this use case we only allow the department node to
//handle the drop. For others we show a user information in a
//popup

List dropRowKey = (List) dropEvent.getDropSite();
//if no dropsite then drop area was not a data area
if(dropRowKey == null){
    return DnDAction.NONE;
}

//make row current so we can access it. This line of code is
//the reason why we needed to keep track of the original tree
//selection state. Here we programmatically change the row
//currency without telling the underlying ADF binding layer.
//If we don't set the row selection back to the original value
//we would risk the binding layer (model) and the client tree
//state to go out of synch
tree.setRowKey(dropRowKey);
JUCtrlHierNodeBinding dropNode =
    (JUCtrlHierNodeBinding) tree.getRowData();
String dropNodeVO =
    dropNode.getHierTypeBinding().getStructureDefName();

//The ADF Binding layer tells us about the structure definition
//name of a node. If you select a node in the binding editor then
//a string like shown below is displayed for each node. Using this
//information you can tell what type the node is of. In our case
//if the type is department, the drop is allowed

if (dropNodeVO.equalsIgnoreCase
    ("adf.sample.model.vo.DepartmentsView")) {

    Row treeRow = dropNode.getRow();
    Transferable t = dropEvent.getTransferable();
    DataFlavor<RowKeySet> df =
        DataFlavor.getDataFlavor(RowKeySet.class, "rowmove");
    RowKeySet rks = t.getData(df);
    Iterator iter = rks.iterator();
```

```
//we allow multi-row drag and drop so that multiple employees
//can be relocated at the same time
while (iter.hasNext()) {
    //get next selected row key
    List key = (List)iter.next();
    table.setRowKey(key);
    //the table model represents its row by the ADF binding class,
    //which is JUCtrlHierNodeBinding

    JUCtrlHierNodeBinding rowBinding =
        (JUCtrlHierNodeBinding)table.getRowData();

    //DCDataRow is the generic row class representation in in ADF.
    //It is the super class of oracle.jbo.Row, which you use with
    //ADF BC services

    Row tableRow = (Row)rowBinding.getRow();

    //to relocate employees, just change the department ID

    tableRow.setAttribute("DepartmentId",
        treeRow.getAttribute("DepartmentId"));
}
//set current row key back
tree.setRowKey(currentRowKey);

//Show the employees in their new department by expanding the
//target department node. For this we need to
// 1. get access to the ADF tree binding
// 2. iteratively get all parent nodes
// 3. get the child nodes (employees) of the drop node

//1. get access to the ADF tree binding. We get this from the
//tree instance we obtained from the dropEvent target
CollectionModel treeModel = (CollectionModel) tree.getValue();
JUCtrlHierBinding treeBinding =
    (JUCtrlHierBinding) treeModel.getWrappedData();

//create a new row key set
RowKeySetImpl rksImpl = new RowKeySetImpl();
rksImpl.add(dropRowKey);

//2. Get all parent nodes node path information to add to the
//row key set to disclose the node
```

```
JUCtrlHierNodeBinding treeDropNode =
    treeBinding.findNodeByKeyPath(dropRowKey);
JUCtrlHierNodeBinding rootNode =
    treeBinding.getRootNodeBinding();
JUCtrlHierNodeBinding dropNodeParent = treeDropNode.getParent();

//walk up the tree to expand all parent nodes
while(dropNodeParent != null && dropNodeParent != rootNode){
    rksImpl.add(dropNodeParent.getKeyPath());
    dropNodeParent = dropNodeParent.getParent();
}

//3. get all employee nodes in a tree to disclose them too
ArrayList<JUCtrlHierNodeBinding> childList =
    (ArrayList<JUCtrlHierNodeBinding>) treeDropNode.getChildren();
for(JUCtrlHierNodeBinding nb : childList){
    rksImpl.add(nb.getKeyPath());
}

//ready to disclose
tree.setDisclosedRowKeys(rksImpl);
AdfFacesContext.getCurrentInstance()
    .addPartialTarget(tree.getParent());

AdfFacesContext.getCurrentInstance().addPartialTarget(table);
return DnDAction.MOVE;
}

//set current row key back
tree.setRowKey(currentRowKey);
//notify the user about failed drop
RichPopup userInfo = (RichPopup) findComponentOnPage("p2");
RichPopup.PopupHints hints = new RichPopup.PopupHints();
userInfo.show(hints);
return DnDAction.NONE;
}

private UIComponent findComponentOnPage(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) {  
  
        lookupComponent = uiComponent;  
    }  
    });  
    return lookupComponent;  
    }  
}
```

Download

You can download the Oracle JDeveloper 11.1.1.6 sample application from ADF Code Corner where it is sample # 101

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

You need to configure the database connection to point to the HR schema in your local database.

RELATED DOCUMENTATION

<input type="checkbox"/>	
<input type="checkbox"/>	
<input type="checkbox"/>	