ADF Code Corner

106. Drag-and-drop reordering of table rows

Abstract:

A requirement drequently posted on the JDeveloper and ADF forum on OTN is to reorder rows exposed in an ADF table.

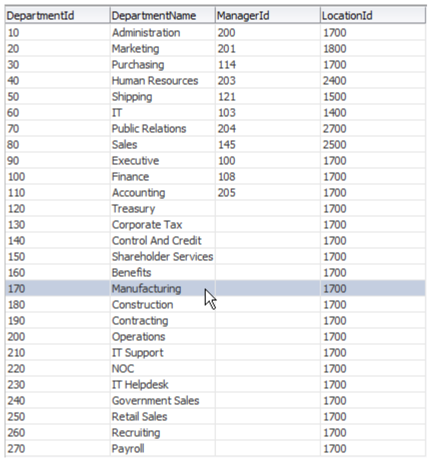
Though you cannot change the order in which existing data is stored in database tables, you can use drag and drop within a table to change the order of displayed rows. A use case for this requirement is to easily compare two rows in a table.

twitter.com/adfcodecorner

Author: Frank Nimphius, Oracle Corporation
twitter.com/fnimphiu
18-MAR-2013

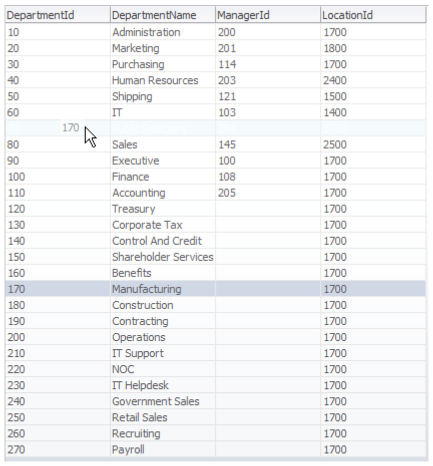
Introduction

As shown in the images below, applying the code in this article will allow you to re-arrange rows within a table using drag and drop.The dragged row is put into the place of the row it is dropped onto. Its simply a change in the indexing of the iterator.



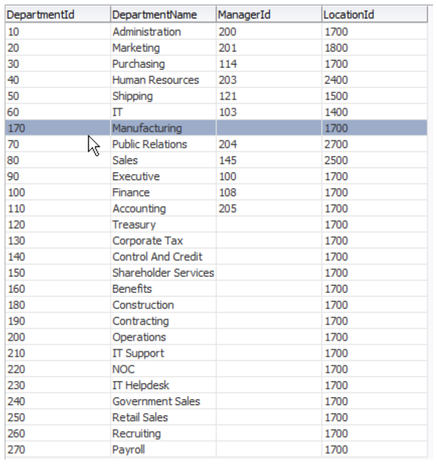
DepartmentId	DepartmentName	ManagerId	LocationId
10	Administration	200	1700
20	Marketing	201	1800
30	Purchasing	114	1700
40	Human Resources	203	2400
50	Shipping	121	1500
60	IT	103	1400
70	Public Relations	204	2700
80	Sales	145	2500
90	Executive	100	1700
100	Finance	108	1700
110	Accounting	205	1700
120	Treasury		1700
130	Corporate Tax		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
220	NOC		1700
230	IT Helpdesk		1700
240	Government Sales		1700
250	Retail Sales		1700
260	Recruiting		1700
270	Payroll		1700

Selecting a row and dragging it to a new location as shown in the images above and below …



DepartmentId	DepartmentName	ManagerId	LocationId
10	Administration	200	1700
20	Marketing	201	1800
30	Purchasing	114	1700
40	Human Resources	203	2400
50	Shipping	121	1500
60	IT	103	1400
170			
80	Sales	145	2500
90	Executive	100	1700
100	Finance	108	1700
110	Accounting	205	1700
120	Treasury		1700
130	Corporate Tax		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
220	NOC		1700
230	IT Helpdesk		1700
240	Government Sales		1700
250	Retail Sales		1700
260	Recruiting		1700
270	Payroll		1700

.. will then show it in its new position within the table. For this to display correctly, the table needs to be partially refreshed, causing a little flicker.



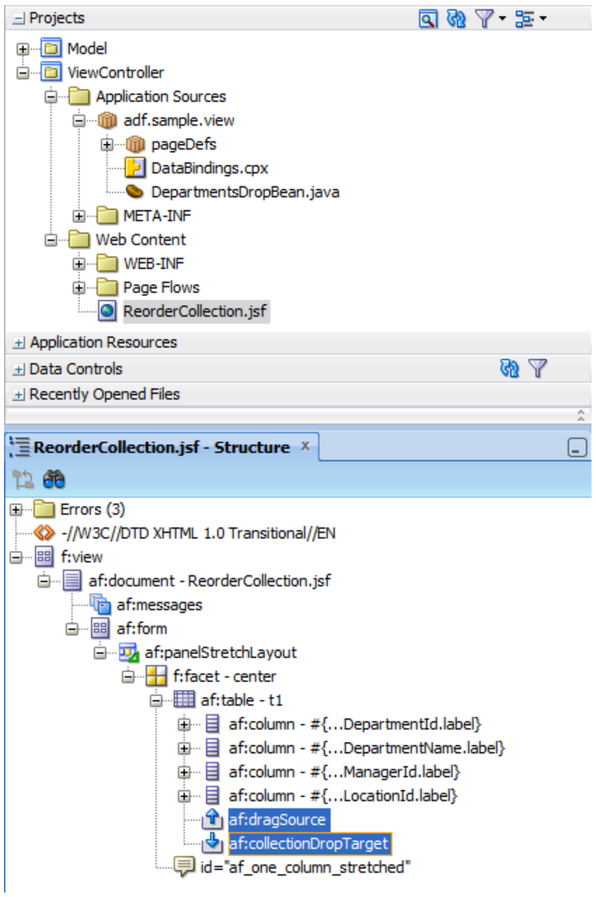
DepartmentId	DepartmentName	ManagerId	LocationId
10	Administration	200	1700
20	Marketing	201	1800
30	Purchasing	114	1700
40	Human Resources	203	2400
50	Shipping	121	1500
60	IT	103	1400
170	Manufacturing		1700
70	Public Relations	204	2700
80	Sales	145	2500
90	Executive	100	1700
100	Finance	108	1700
110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

Note that reordering the rows in the table will only have an effect to the DCIteratorBinding (more precise, its an effect to the RowSetIterator it decorates) in the Pagedef file. The same data in the database doesn't change and performing a re-query of the iterator data will produce the previous row order unless you dynamically change the query order according to the drag and drop of rows (which is not in the scope of this article)

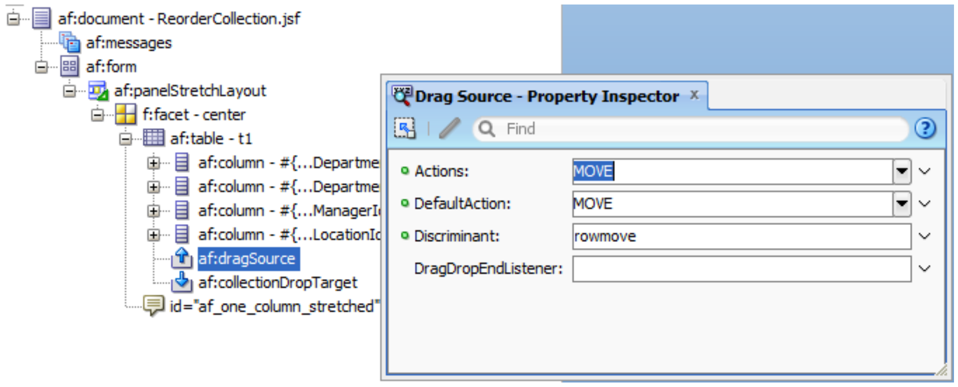
Implementation

The image below shows the JDeveloper 11.1.2.3 workspace of the sample (Note though that this solution also works with 11g R1 applications).

The row reordering for a table is built out of the `af:dragSource` tag that enables rows to be dragged within a table, the `af:collectionDropTarget` tag that takes the drop event and code in a managed bean that performs the change in the iterator index.



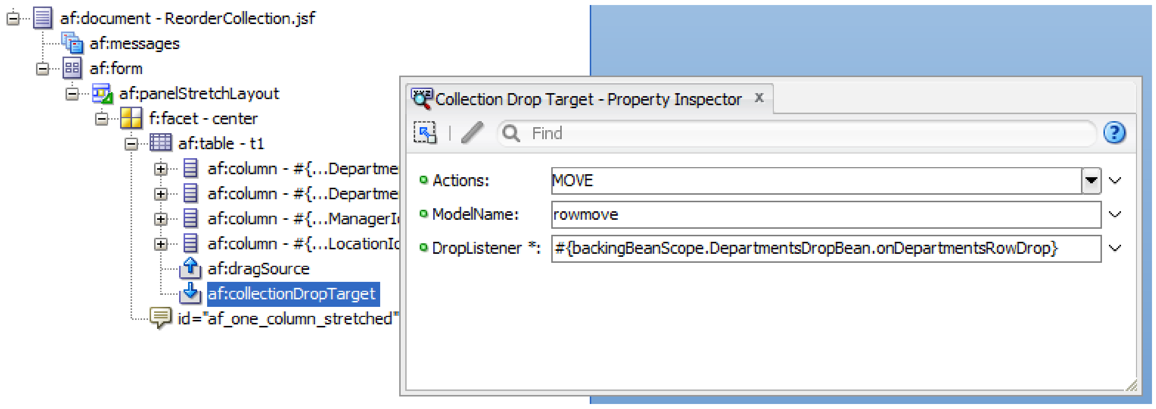
The image below shows the configuration of the **dragSource** tag, which consists of an Action, MOVE, and a **Discimimant**, "rowmove". The Discriminant is used to help the drag source and drop target to find together, which not only is required but also useful if you wanted to have multiple drag and drop implementations on a single view.


af:document - ReorderCollection.jsf
af:messages
af:form
af:panelStretchLayout
f:facet - center
af:table - t1
af:column - #{...Departme
af:column - #{...Departme
af:column - #{...ManagerI
af:column - #{...LocationIc
af:dragSource
af:collectionDropTarget
id="af_one_column_stretched"
Drag Source - Property Inspector
Find

Actions:	MOVE
DefaultAction:	MOVE
Discriminant:	rowmove
DragDropEndListener:	

The **collectionDropTarget** tag is configured as shown in the image below. For the drop component, the Action is set to MOVE as well (at least one action needs to be chosen that matches the action defined on the drag source) and the **ModelName** is set to **rowmove**. It is kind of unfortunate that the discriminator property naming is not consistent in ADF Faces, but at least they work as designed ;-)

The **DropListener** property references a managed bean in backing bean or request scope. The bean doesn't hold any state and therefore doesn't need to survive a single request.


af:document - ReorderCollection.jsf
af:messages
af:form
af:panelStretchLayout
f:facet - center
af:table - t1
af:column - #{...Departme
af:column - #{...Departme
af:column - #{...ManagerI
af:column - #{...LocationI
af:dragSource
af:collectionDropTarget
id="af_one_column_stretched"
Collection Drop Target - Property Inspector
Find

Actions: MOVE
ModelName: rowmove
DropListener *: #{backingBeanScope.DepartmentsDropBean.onDepartmentsRowDrop}

The signature for the managed bean method is as follows …

```
public DnDAction methodName(DropEvent eventName)
```

… and is created automatically when using the arrow-down -> Edit option to built the method.

Below is the listing of the commented source code of the managed bean that handles the drop for you to study and modify so it serves your use case.

DepartmentsDropBean Managed Bean

```
import java.util.Iterator;
import java.util.List;

import oracle.adf.model.binding.DCIteratorBinding;
import oracle.adf.view.rich.component.rich.data.RichTable;
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.RowSetIterator;
import oracle.jbo.uicli.binding.JUCtrlHierBinding;
import oracle.jbo.uicli.binding.JUCtrlHierNodeBinding;

import org.apache.myfaces.trinidad.model.CollectionModel;
import org.apache.myfaces.trinidad.model.RowKeySet;

public class DepartmentsDropBean {
  public DepartmentsDropBean() {}

 /**
   * The dragged row takes the position of the row it is droped on.
   * Note that drag and drop is on the table component and to
   * synchronize the underlying ADF model I needed to accept a
   * "flicker" of the table after the drop
   *
   * @param dropEvent Event passed in from ADF Faces at the end of the
   * drag and drop operation
   * @return Copy.MOVE
 */
public DnDAction onDepartmentsRowDrop(DropEvent dropEvent) {
 //get the table instance. This information is later used
 //to determine the tree binding and the iterator binding
 RichTable table = (RichTable)dropEvent.getDragComponent();

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

 //The transferable is the payload that contains the dragged row's
 //row key that we use to access the dragged row handle in the ADF
 //iterator binding
 Transferable t = dropEvent.getTransferable();

 //get the row key set of the dragged row. The "rowmove" string is the
 //discriminant defined on the drag source and the collectionDrop
 //target.
 DataFlavor<RowKeySet> df = DataFlavor.getDataFlavor
                                       (RowKeySet.class, "rowmove");
 RowKeySet rks = t.getData(df);
 Iterator iter = rks.iterator();


 //for this use case the re-order of rows is one-by-one, which means
 //that the rowKeySet  should only contain a single entry. If it
 //contains more then still we only look at a singe (first) row key
 //entry

 List draggedRowKey = (List) iter.next();

 //get access to the oracle.jbo.Row instance representing this table
 //row
 JUCtrlHierNodeBinding draggeRowNode =
          (JUCtrlHierNodeBinding) table.getRowData(draggedRowKey);
 Row dragRow = draggeRowNode.getRow();

 JUCtrlHierNodeBinding dropRowObject =
            (JUCtrlHierNodeBinding) table.getRowData(dropRowKey);
 Row dropRow = dropRowObject.getRow();
```

```
  //get the table's ADF JUCtrlHierBinding
  CollectionModel collectionModel =
                                (CollectionModel) table.getValue();
  JUCtrlHierBinding treeBinding =
                (JUCtrlHierBinding)collectionModel.getWrappedData();

  //get access to the ADF iterator binding used by the table and the
  //underlying RowSetIterator. The RowSetIterator allows us to remove
  //and re-instert the dragged row

  DCIteratorBinding departmentsIterator =
                                 treeBinding.getDCIteratorBinding();
  RowSetIterator rsi = departmentsIterator.getRowSetIterator();
  int indexOfDropRow= rsi.getRangeIndexOf(dropRow);
  //remove dragged row from collection so it can be added back
  dragRow.removeAndRetain();

  rsi.insertRowAtRangeIndex(indexOfDropRow, dragRow);
  //make row current in ADF iterator.
  departmentsIterator.setCurrentRowIndexInRange(indexOfDropRow);

  //ppr the table
  AdfFacesContext adfctx = AdfFacesContext.getCurrentInstance();

  //note that the refresh of the table didn't work when refreshing the
  //table so I needed to refresh the container component
  //(af:panelStretchLayout).

  adfctx.addPartialTarget(table.getParent());
  return DnDAction.MOVE;
 }
}
```

Download

The sample application for this article can be downloaded as sample #106 from the ADF Code Corner website

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

You need to configure the database connection of the model project to point to the unlocked HR schema of a local database. Then run the "ReorderCollection.jsf" to play with this.

RELATED DOCOMENTATION

⊠	
⊠	
