ADF Code Corner

100. How-to undo table row selection in case of custom validation failure

twitter.com/adfcodecorner

Abstract:

A question on OTN was about how to set back the table row selection in an ADF table if business logic validation fails. In the particular use case, the poster on OTN wanted to verify that a data change in a table was committed before the user could move on to a next row in the table. This article provides a solution for the very same use case

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

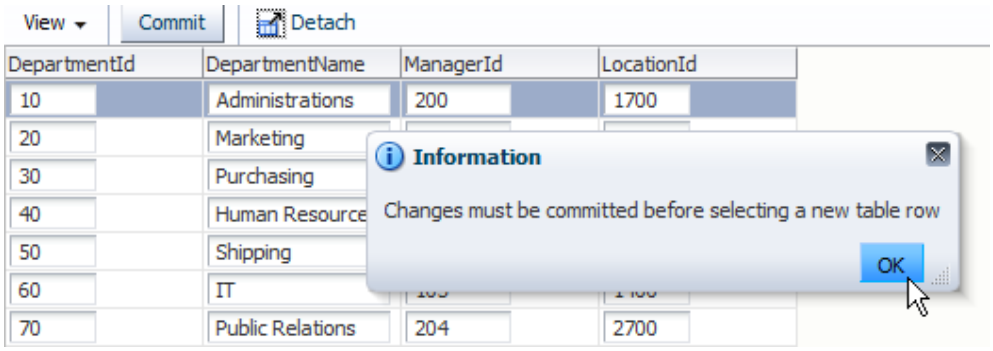
Please post questions or report problems related to the samples in this series on the OTN forum for Oracle JDeveloper: http://forums.oracle.com/forums/forum.jspa?forumID=83

Introduction

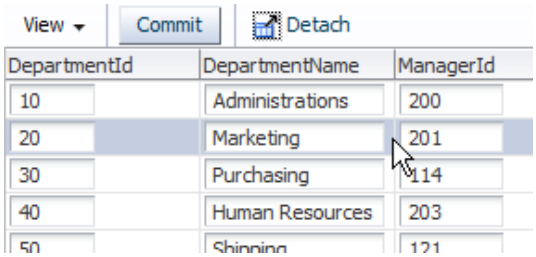
The sample application that you can download from the ADF Code Corner website shows a single editable table. Each cell in the table performs an autosubmit in response to a value change issued by the application user. This way the transaction state of the application is "dirtied" when row data gets changed. Changing data in a table row and clicking into another table row will do two things

1. An alert is shown notifying the user that commit has ot be called (or rollback to give him or her a choice)

2. The original table row selection is restored

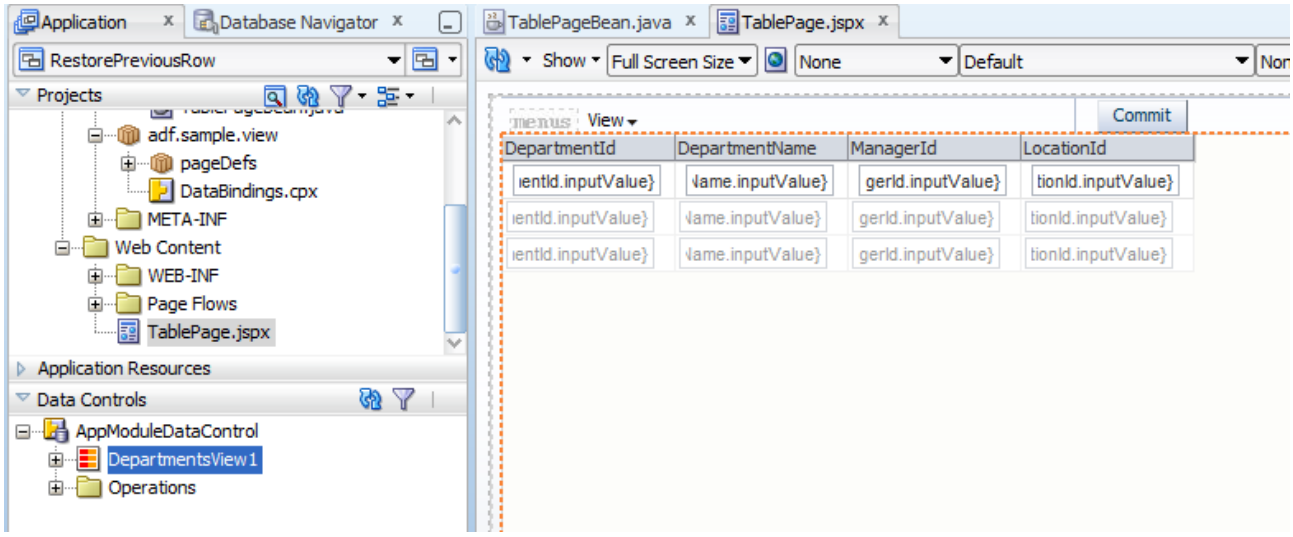


If no pending data is detected when the user changes the table row currency, then the selected ADF Faces table row is set as the current row in the ADF binding layer (which almost indicates how the solution to this challenge works)



Implementing the solution

The editable table is created by dragging a collection from the data controls panel to the JSF page and choosing **ADF Form** from the choices displayed in the ADF context menu. In the sample, the table is placed into an `af:panelCollection` mainly for the reason to have a place to put the commit button to. The commit button is created by dragging the ADF Business Components **commit** operation from the Data Controls panel on top of a command tool bar button added to the panel collection toolbar area.

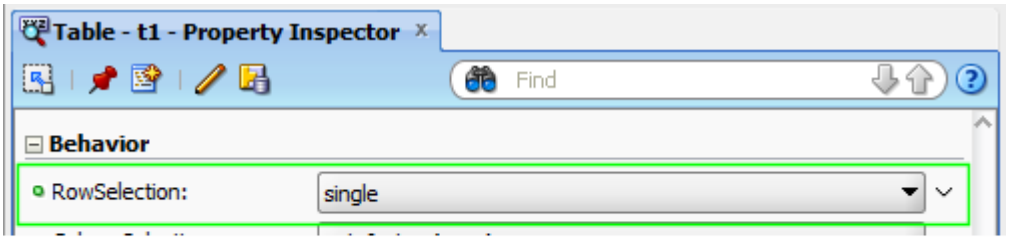


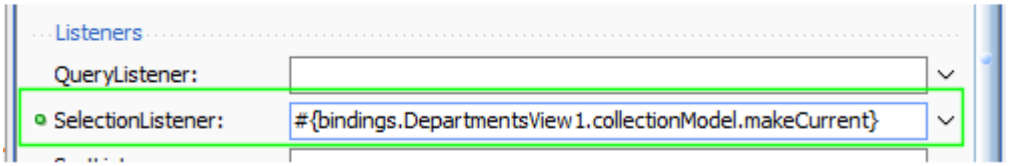
There are three properties of the table that are important for the solution to the use case reported on OTN.

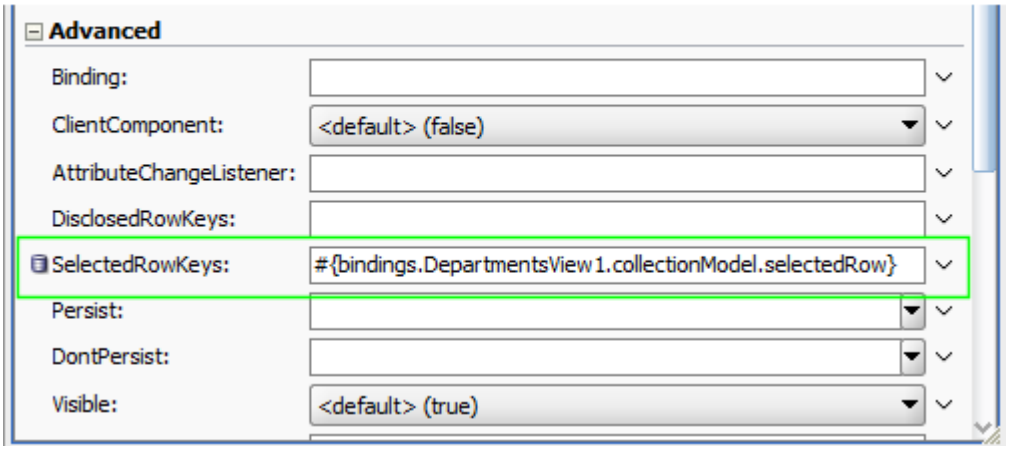
RowSelection – the sample handles the single row selection case. Though the code can be changed to work in a multi row select case, it makes sense to have RowSelection set to *single*.

SelectionListener – the SelectionListener property references the ADF binding layer to ensure the selected ADF Faces table row is set as the current row in the underlying ADF tree binding (tree bindings are used for binding table data as well). Without this listener the ADF Faces table component and the ADF binding layer would go out of synch.

SelectedRowKeys – the SelectedRowKey property references a collection of row keys that should be set as selected. In a single select scenario, this collection only has a single entry.

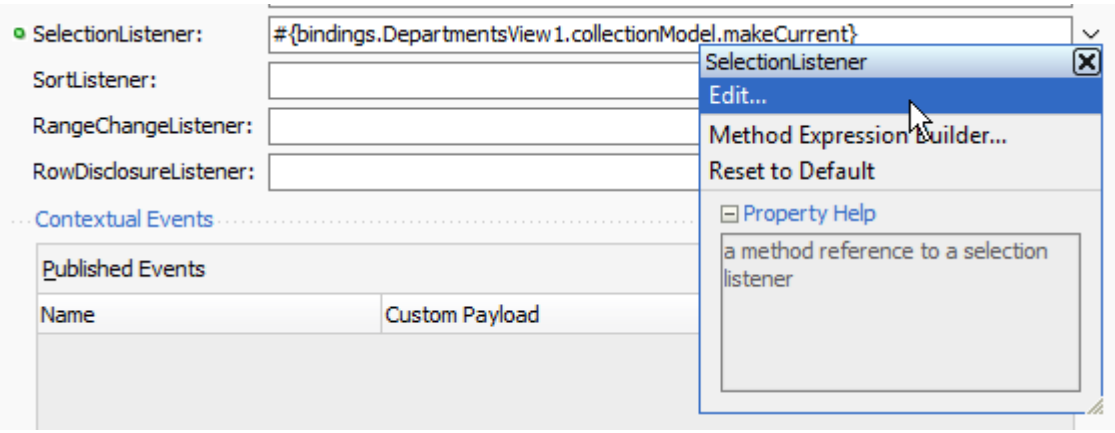




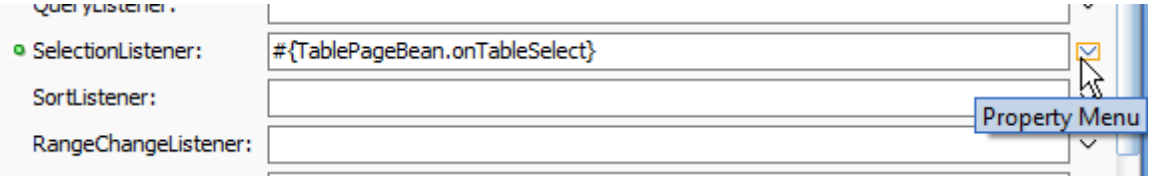


As mentioned, the **SelectionListener** ensures the table row select state is synchronized with the current row setting of the ADF binding layer. To be able to intercept the user row selection, perform a custom validation and then either to continue or set the row currency back to the previous selection, we need to decouple the table row selection from the ADF binding.

The way to do this is to create a custom select listener and reference it from the **SelectionListener** property. To create a custom listener, click the "down arrow" icon at the end of the **SelectionListener** property field and choose **Edit** from the context menu.



In the opened dialog you can either create a new managed bean to hold the listener, or choose an existing one. With the new configuration, all table row selection will divert to the managed bean.



Below is the commented managed bean code. The managed bean performs the following tasks

1. Read the previous selected row key from the event object so the "old" state can be restored
2. Get access to the ADF binding layer through the table's value property pointing to the `CollectionModel`. The binding layer (`JUCtrlHierBinding`) is used to synchronize the current row of the binding layer with the current table selection for the case in which the user is allowed to change the table row selection.
3. Access the `ControllerContext` and its current ViewPort. The current view port either is a browser view, or a region displaying a page fragment. The ViewPort allows the managed bean to check if the transaction for the current task flow (unbounded or bounded) is dirty, in which case uncommitted data exist.
4. Show an alert if uncommitted data exist and set the ADF Faces table **SelectedRowKeySet** to the old selection state saved in the first step.
5. If no uncommitted data exist, synchronize the ADF binding layer with the current ADF Faces table selection. This substitutes the previous #{bindings.<tree binding id>.makeCurrent} EL configuration of the SelectionListener property.

```
import javax.faces.application.FacesMessage;
import javax.faces.context.FacesContext;
import oracle.adf.controller.ControllerContext;

import oracle.adf.view.rich.component.rich.data.RichTable;
import oracle.adf.view.rich.context.AdfFacesContext;
```

```
import oracle.jbo.Row;
import oracle.adf.model.binding.DCIteratorBinding;
import oracle.jbo.uicli.binding.JUCtrlHierBinding;
import oracle.jbo.uicli.binding.JUCtrlHierNodeBinding;

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


public class TablePageBean {
    public TablePageBean() {
}



/**
   * Handle single row selection use case. If the previous row
   * contains uncommitted data, undo row selection and inform user
   * that he/she should commit a value change before editing the next
   * row
   *  @param selectionEvent
*/
public void onTableSelect(SelectionEvent selectionEvent) {
  //keep track of (old) selected row key
   RowKeySet oldKeySet = selectionEvent.getRemovedSet();

   //until here only the row selection in the table has
   //changed. The ADF binding layer doesn't know about the row
   //selection yet. If there is uncommitted data then we don't
   //change current row on the binding and just set the table key
   //back

   RichTable table = (RichTable)selectionEvent.getSource();
   //From the table, get the associated ADF tree binding. This handle
   //is used later to set the current row in the binding to the row
   //selected in the ADF Faces table.

   CollectionModel tableModel = (CollectionModel)table.getValue();
   JUCtrlHierBinding adfTableBinding =
      (JUCtrlHierBinding)tableModel.getWrappedData();
   ControllerContext cctx = ControllerContext.getInstance();
       if(cctx.getCurrentRootViewPort().isDataDirty()){
   FacesMessage message = new FacesMessage(FacesMessage.SEVERITY_INFO,
        "Please Commit Changes",
         "Changes must be committed before selecting a new table row");
   FacesContext fctx = FacesContext.getCurrentInstance();
```

```
    fctx.addMessage(null, message);

    table.setSelectedRowKeys(oldKeySet);
    AdfFacesContext adfFacesCtx = AdfFacesContext.getCurrentInstance();

   //refresh table to show current selected row set back to
   //original row. Note that though we set the row back, no row
   //selection event is fired. Row selection events fire only if
   //users select a row

   adfFacesCtx.addPartialTarget(table);
   fctx.renderResponse();
}
else{
    //Next: Make the selected table row the current row in the
    //ADF binding layer. The code below is the substitution of the
    // default SelectionListener definition JDeveloper set to
    //#{bindings.DepartmentsVie1.makeCurrent} or, put in more generic
    //terms, #{bindings.<tree binding Id>.makeCurrent}

    JUCtrlHierNodeBinding tableRowBinding =
              (JUCtrlHierNodeBinding) table.getSelectedRowData();
    Row row = tableRowBinding.getRow();
    //access the iterator used by the table binding
    DCIteratorBinding iter = adfTableBinding.getDCIteratorBinding(;)
    iter.getRowSetIterator().setCurrentRow(row);

    //no need to refresh the ADF Faces table because table and binding
    //are in synch
   }
  }

}
```

Note: The sample code above calls `getCurrentRootViewPort` on the controller context to check for the transaction state. If the code needs to also work for bounded task flows configured to have their own transaction in an ADF region, you use **`getCurrentViewPort`** instead.

Sample Download

The Oracle JDeveloper 11g R1 (11.1.1.6) sample application explained in this article can be downloaded as sample #100 from the ADF Code Corner web site

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

You need to configure the database connect for the application to point to the HR schema of your Oracle database installation.