Creating new row of view object in ADF | Techartifact

**Requirment**– Creating new rows through the view object

**Solution**-Create a new view row by calling createRow() on the view object as shown in the following code snippet:

|  |
| --- |
| /\*\*The createEmployee is a custom method defined in application  \* module implementation class.  \*/      public void createEmployee () {  //Get the view object  ViewObject employee= findViewObject("EmployeeVO");  // Create a row and fill in the columns.  Row row = employee.createRow();  row.setAttribute("Name", "Vinay");  row.setAttribute("EmpId", 1);  //Insert row in to the default row set  employee.insertRow(row);  } |

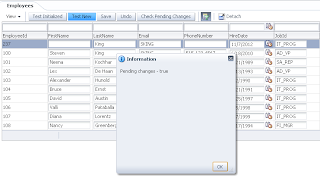
You can also use createAndInitRow(AttributeList initVals) on the view object to create a new view row. This API allows you to initialize the row with an attribute list.

### [Difference Between Initialized and New Mode in ADF BC](http://andrejusb.blogspot.com/2012/11/difference-between-initialized-and-new.html)

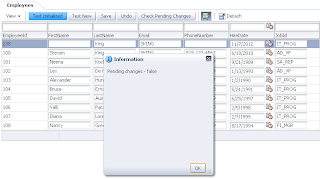
Its always a bit confusing - what is actually the difference between Initialized and New mode for created rows in ADF BC. I have found good information in ADF Developer Guide for this - [27.4.5 What You May Need to Know About Create and CreateInsert](http://docs.oracle.com/cd/E35521_01/web.111230/e16182/web_tables_forms.htm#ADFFD744). In short - when you create new row, it gets status New. If there is requirement to deregister created (but not yet inserted into DB) from pending changes list - you can set Initialized status for such row. It will keep Initialized status, until user will change attribute value.  
  
I have created small sample application to test this behavior - [NewInitializedRowADF.zip](http://jdevsamples.googlecode.com/files/NewInitializedRowADFApp.zip). This application implementes two methods in AM implementation class:

[](http://2.bp.blogspot.com/-ZG0V5KNHNVU/UJv1HEFXpmI/AAAAAAAAIA0/_RXpONsGJlw/s1600/3.png)

These two methods are almost the same, except - row.setNewRowState(Row.STATUS\_INITIALIZED) statement in the first method. This methods assigns newly created row with Initialized status. Both methods are invoked from ADF UI. Sample application is based on Employees table from HR schema. This table contains number of required attributes - for the test purpose, I have set default values for each of required attributes (this allows to avoid validation errors).  
  
Test is really simple - calling one method and then checking if pending changes exist, same done for the second method.  
  
Firstly I was testing the method where row is created with the default status New. Pending changes are reported:

[](http://2.bp.blogspot.com/-6hM91rL8XBw/UJv3XN0QZVI/AAAAAAAAIA8/MDXiN8-Y3_E/s1600/1.png)

Next - method with row.setNewRowState(Row.STATUS\_INITIALIZED) statement was tested. No pending changes detected for inserted row:

[](http://3.bp.blogspot.com/-bBGP-vp9-UE/UJv3r7nx5mI/AAAAAAAAIBE/PShF8m12ce4/s1600/2.png)

Setting row with Initialized status is useful, when we want to delay pending changes check. For example, when new row is created with default values from PL/SQL functions - we don't want to consider it as candidate for pending changes in ADF, until user types value by himself.

Posted by Andrejus Baranovskis at [7:21 PM](http://andrejusb.blogspot.com/2012/11/difference-between-initialized-and-new.html)

[Email This](https://www.blogger.com/share-post.g?blogID=5874979429188093780&postID=5054050904172669718&target=email)[BlogThis!](https://www.blogger.com/share-post.g?blogID=5874979429188093780&postID=5054050904172669718&target=blog)[Share to Twitter](https://www.blogger.com/share-post.g?blogID=5874979429188093780&postID=5054050904172669718&target=twitter)[Share to Facebook](https://www.blogger.com/share-post.g?blogID=5874979429188093780&postID=5054050904172669718&target=facebook)[Share to Pinterest](https://www.blogger.com/share-post.g?blogID=5874979429188093780&postID=5054050904172669718&target=pinterest)

**Preconditions:**   
Oracle JDeveloper 11.1.1.6.0

I am using ADF Faces and ADF Binding layer to display a list of Java POJO in an af:table. The ADF table is displayed on a view inside a bounded task flow.   
This task flow is used in a dynamic region on another view.   
Navigation contains two links which are used to choose via bean-property which bounded task flow to display in the dynamic region.   
The dynamic region receives an „Input Parameter Map“ via task flow binding.   
This map contains the POJO to display in the table and a state-Bean which should hold table state and state of possible other components located on the view (see implementation below).

**Problem/ requirements:**

**1. Preselection:**

Preselect a single row in the table according to data contained in the POJO on first load of the table.

**2. Keep selection over taskflow switch:**

The selected row in the table (bounded taskflow A) should also be selected after switching to another bounded taskflow B and back to A (where the table is located).

**3. No MDS**

No usage of savepoints/ MDS features.

**4. Consistent table model and binding layer**

It should be possible to ask the table component and the binding for the selected row after switching between the task flows and after selecting elements in the table.   
The table row and the selected row in binding should be consistent after switching the task flows and after selecting elements in the table.

**Already tried:**

Preselection of a row and keep selection over task flow switch without use of MDS/ savepoints.

I have bound some table properties to a “state” bean. Implementation of the bean is attached below.

**1. Properties of the table:**

**Selected row keys:**  
selectedRowKeys="#{pageFlowScope.stateBean.tableStateBean.rowKeySet}"

**Backing-Bean property to bind RichTable:**   
binding="#{pageFlowScope.stateBean.tableStateBean.richTable}"

**Default selection Listener**   
selectionListener="#{bindings.postanschriften.collectionModel.makeCurrent}"

**What we tried here:**   
On initial load of the table and changing the row bean method getRowKeySet(..) is called.   
In this method the row to select is calculated on initial table load (this.rowKeySet == null). Because table property selectedRowKeys is bound to pageFlowScope.stateBean.tableStateBean.rowKeySet the property in the bean is updated if another row is selected in table. If we change the task flow from A to B and vice versa the getter of backing bean property richTable is called. In the getter method the selection state is restored.

**2. Implementation of the “state” bean:**

public class TableStateBean {

private RichTable richTable;

private RowKeySet rowKeySet;

private String bindingIteratorName;

private RowMatcher matcher;

public RowKeySet getRowKeySet() {

if (this.rowKeySet == null) {

preselectRow();

}

return rowKeySet;

}

public RichTable getRichTable() {

if (richTable != null && rowKeySet != null) {

RowKeySet currentTableSelection = getCurrentTableSelection();

RowKeySet tableSelectionToRestore = getTableSelectionToRestore();

executeSelection(tableSelectionToRestore, currentTableSelection);

}

return richTable;

}

public void setRichTable(RichTable pRichTable) {

richTable = pRichTable;

}

public void doTableSelection(){

preselectRow();

}

private RowKeySet getCurrentTableSelection() {

Row currentRow = (Row) JSFUtils.resolveExpression("#{bindings." + this.bindingIteratorName + ".currentRow}");

Key currKey = currentRow.getKey();

ArrayList<Key> lst = new ArrayList<Key>(1);

lst.add(currKey);

RowKeySet keySet = new RowKeySetImpl();

keySet.add(lst);

return keySet;

}

private RowKeySet getTableSelectionToRestore() {

RowKeySet tableSelectionToRestoreRow = null;

RowSetIterator rowSetIterator = extractRowSetIterator();

int tableRowIndexOfCurrentSelectedKey = getSingleRowKeyIndexValue(this.rowKeySet);

if (tableRowIndexOfCurrentSelectedKey != -1) {

Row currentRow = rowSetIterator.first();

for (int i = 0; rowSetIterator.hasNext() && i < tableRowIndexOfCurrentSelectedKey; i++) {

currentRow = rowSetIterator.next();

}

if (currentRow != null) {

Key newSelectionKey = currentRow.getKey();

ArrayList<Key> keyList = new ArrayList<Key>(1);

keyList.add(newSelectionKey);

tableSelectionToRestoreRow = new RowKeySetImpl();

tableSelectionToRestoreRow.add(keyList);

}

}

return tableSelectionToRestoreRow;

}

private int getSingleRowKeyIndexValue(RowKeySet rowKeySet) {

int tableRowIndexOfCurrentSelectedKey = -1;

if (rowKeySet != null) {

Object[] rowKeySetArray = rowKeySet.toArray();

List<Key> selectedRowKeys = (rowKeySetArray.length > 0) ? (List<Key>) rowKeySetArray[0] : new ArrayList<Key>();

if (selectedRowKeys.size() > 0) {

Key currentSelectedKey = selectedRowKeys.get(0);

Object[] attributeValues = currentSelectedKey.getAttributeValues();

assert (attributeValues.length > 0);

tableRowIndexOfCurrentSelectedKey = (Integer) attributeValues[0];

}

}

return tableRowIndexOfCurrentSelectedKey;

}

private void executeSelection(RowKeySet newCurrentRow, RowKeySet oldCurrentRow) {

SelectionEvent selectionEvent = new SelectionEvent(oldCurrentRow, newCurrentRow, richTable);

selectionEvent.queue();

AdfFacesContext.getCurrentInstance().addPartialTarget(richTable);

}

protected void preselectRow() {

RowSetIterator rowSetIterator = extractRowSetIterator();

RowKeySet oldSelection = getCurrentTableSelection();

Row currentRow = rowSetIterator.first();

while (rowSetIterator.hasNext()

&& (!matcher.match(currentRow))) // Matcher selects which row should be displayed according to the Object bound behind the binding-layer.

{

currentRow = rowSetIterator.next();

}

if (currentRow != null) {

Key key = currentRow.getKey();

RowKeySet newSelection = createRowKeySet(key);

setActiveRowKey(key);

executeSelection(newSelection, oldSelection);

setRowKeySet(newSelection);

}

}

private void setActiveRowKey(Key pKey) {

ArrayList<Key> lst = new ArrayList<Key>(1);

lst.add(pKey);

this.richTable.setActiveRowKey(lst);

}

private RowKeySet createRowKeySet(Key pKey) {

ArrayList<Key> lst = new ArrayList<Key>(1);

lst.add(pKey);

RowKeySetImpl rowKeySetToCreate = new RowKeySetImpl();

rowKeySetToCreate.add(lst);

return rowKeySetToCreate;

}

private RowSetIterator extractRowSetIterator() {

DCIteratorBinding iteratorBinding = ADFUtils.findIterator(this.bindingIteratorName);

RowSetIterator rowSetIterator = iteratorBinding.getRowSetIterator();

return rowSetIterator;

}

}

It seems that in some cases the binding layer is not synchronous to the elements selected on the RichTable component. So I guess the solution mentioned above is not very robust.

Is there another way to archive the functionality mentioned above in a robust manner? I think to preselect a row in a table according to some values in the bound POJO is not so strange. And also keeping the selected row of a table after switching between bounded task flows (dynamic region) should be possible, isn´t it?

Thank you in anticipation for your help

Regards,

Max

[jsf](https://stackoverflow.com/questions/tagged/jsf) [oracle-adf](https://stackoverflow.com/questions/tagged/oracle-adf)

[share](https://stackoverflow.com/q/13469113)[improve this question](https://stackoverflow.com/posts/13469113/edit)

[edited Nov 8 '15 at 8:21](https://stackoverflow.com/posts/13469113/revisions)

[[https://www.gravatar.com/avatar/89927e2f4bde24991649b353a37678b9?s=32&d=identicon&r=PG](https://stackoverflow.com/users/157882/balusc)](https://stackoverflow.com/users/157882/balusc)

[BalusC](https://stackoverflow.com/users/157882/balusc)

**847k**29831423217

asked Nov 20 '12 at 8:22

[[https://i.stack.imgur.com/sfkco.jpg?s=32&g=1](https://stackoverflow.com/users/1513980/max)](https://stackoverflow.com/users/1513980/max)

[Max](https://stackoverflow.com/users/1513980/max)

**200**618

add a comment

2 Answers

[active](https://stackoverflow.com/questions/13469113/adf-aftable-restoring-table-state?answertab=active#tab-top)[oldest](https://stackoverflow.com/questions/13469113/adf-aftable-restoring-table-state?answertab=oldest#tab-top)[votes](https://stackoverflow.com/questions/13469113/adf-aftable-restoring-table-state?answertab=votes#tab-top)

0

Setting a 'selected' row is rather easy. There are several solutions for that. This is just one of them : Make a binding for your table to you backing bean. In the getter you add the following code:

DCBindingContainer bindings = (DCBindingContainer)BindingContext.getCurrent().getCurrentBindingsEntry();

DCIteratorBinding dcItteratorBindings =

bindings.findIteratorBinding("yourViewObjectIterator");

RowSetIterator it = dcItteratorBindings.getRowSetIterator();

Rows[] rows = voTableData.getAllRowsInRange();

Row needsSelection = null;

for(Row r: rows)

{

//just search for the row you want to set selected

if(r.getAttribute("SomeAttribute").eqauls(..))

needsSelection = r;

}

if(needsSelection != null)

it.setCurrentRow(needsSelection);