ADF Code Corner

006. How to cancel an edit form, undoing changes in Java

twitter.com/adfcodecorner

Abstract:

This how-to document describes one of the two options available to cancel an edit form in ADF Faces RC without a required field message being raised by the client validator. Canceling an edit form with ADF requires more than just setting the immediate property set to true on the command button. It requires some housekeeping for the changes performed on the ADF binding layer.

Author: Frank Nimphius, Oracle Corporation
twitter.com/fnimphiu
23-DEC-2008

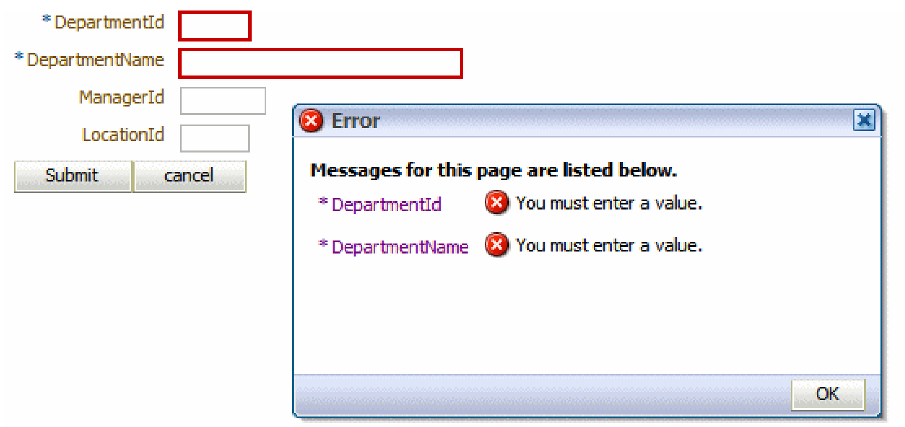
Introduction

Two strategies exist to implement a cancel button for an ADF Faces / ADF edit form that uses client-side and server side validation: a) restoring state with a savepoint set in ADFm and b) a client Java method exposed on the ADF Business Component business service. Both strategies are similar in that they use the *immediate* property on a ADF Faces command button to cancel the edit, but are different in their implementation of the data clean-up. When discussing the problem with Steve Muench, he came up with the the declarative approach that uses a bounded taskflow to clean up the data modification when cancel is pressed. Its a straight forward approach that is easy understand and implement. Especially when re-use is not a requirement, the declarative feature might be seen to complex for such a little functionality, which is why in this how-to document we document the simpler Java version as well.

Note: Another, possibly more elegant form using save points is documented in another article on ADF Code Corner "How-to cancel an edit form, undoing changes with ADFm save points"

Problem Statement

The problem is based on a simple usecase: A user enters a page to either create a new record or update an existing record.



Messages for this page are listed below.
* DepartmentId You must enter a value.
* DepartmentName You must enter a value.

After navigating to the edit form, the user starts modifying the form data and then decides to forget about the changes. So he presses the cancel button of the edit form in which case the ADF Faces RC response might be as shown below

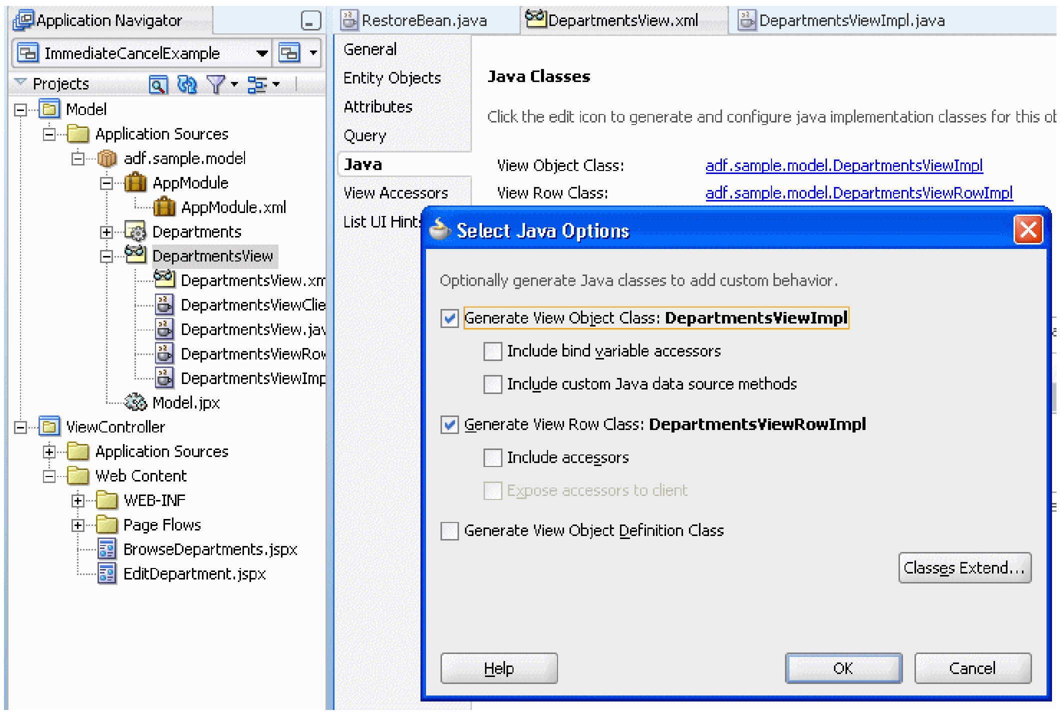
There are two issues here: i) The user didn't provided all the required field values before pressing "cancel" and ii) a new record has been created in the ADF iterator binding, which awaits clean up.

Code Centric Solution

The coded solution includes changes on the ADF Business Component model, the binding layer and the view layer.

ADF Business Components

Since the edit form is used for new records and existing records, we need to know about the row state. If the edited row is new then the clean-up process needs to remove it from the ADF binding and if it is an existing row, we want to refresh it to its old state. The row information is exposed on the *RowImpl* class of the ViewObject. To keep the business service logic and the view layer logic separate, we will create a client method in the *ViewObjectImpl* class that accesses the *RowImpl* class to determine the row status before returning the row status as a String. To create the *RowImpl* and *Impl* classes, open the ViewObject editor for the ViewObject that is used by the edit form. You open the editor by either a double click on the ViewObject entry in the Oracle JDeveloper Application Navigator, or by choosing "Open ..." from the context menu.



Click the edit icon to generate and configure java implementation classes for this ol

View Object Class: adf.sample.model.DepartmentsViewImpl
View Row Class: adf.sample.model.DepartmentsViewRowImpl

Optionally generate Java classes to add custom behavior.

[✓] Generate View Object Class: **DepartmentsViewImpl**
[ ] Include bind variable accessors
[ ] Include custom Java data source methods
[✓] Generate View Row Class: **DepartmentsViewRowImpl**
[ ] Include accessors
[ ] Expose accessors to client
[ ] Generate View Object Definition Class

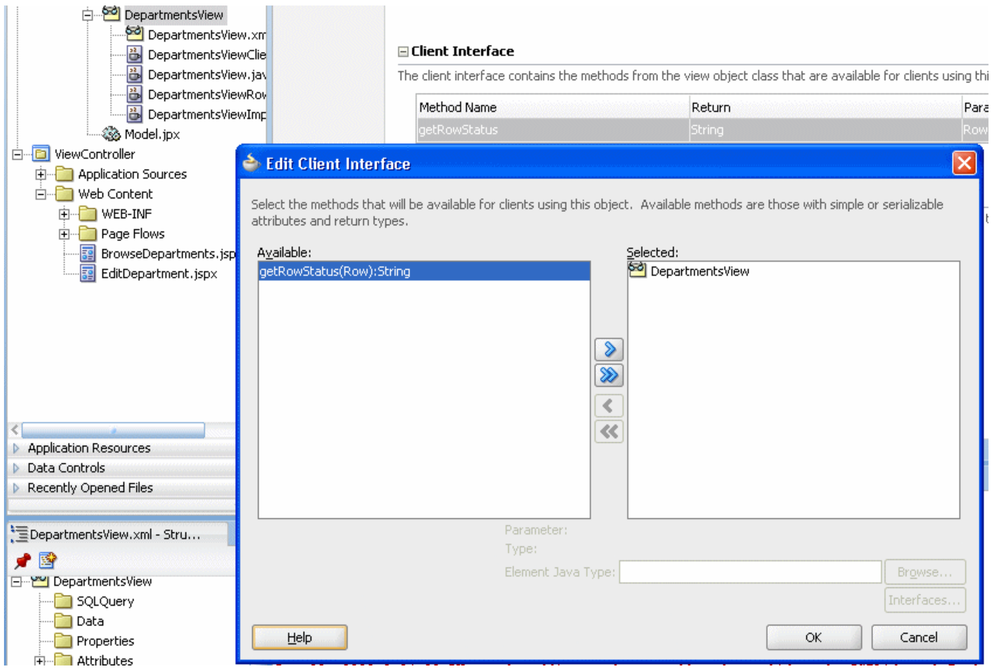
Mark the two checkboxes that initiate the creation of the *ViewImpl* and *ViewRowImpl* class and press **OK**. A less "quick and dirty" solution would be to add the required *ViewImpl* code into a custom *ViewObjectImpl* super class and to extend from this class so the method can be reused with other ViewObjects. In this how-to however, we pretend to only need the row status information for the "DepartmentsView".

Open the *DepartmetsViewImpl* class in the Java code editor and add the following code:

```
public String getRowStatus(Row row){
  DepartmentsViewRowImpl rwImpl = (DepartmentsViewRowImpl)row;
  String rwStatus =
        translateStatusToString(rwImpl.getEntity(0).getEntityState());
  return rwStatus;
}

private String translateStatusToString(byte b) {
  String ret = null;
    switch (b) {
    case Entity.STATUS_INITIALIZED: {
      ret = "Initialized";
     break;
    }
    case Entity.STATUS_MODIFIED: {
      ret = "Modified";
      break;
    }
    case Entity.STATUS_UNMODIFIED: {
      ret = "Unmodified";
      break;
    }
    case Entity.STATUS_NEW: {
      ret = "New";
      break;
    }
   }
return ret;
}
```

The longer part of the code is to translate the row state, which is read from the entity, from byte to String. The *getRowStatus* method is defined as public so it can be exposed on the ViewObject as a client method. For this, open the ViewObject and select the "Java" node. Press the pencil icon on the "Client Interface" section and shuttle the method from the left side to the right.



The client interface contains the methods from the view object class that are available for clients using thi

Method Name	Return	Para
getRowStatus	String	Row

Edit Client Interface

Select the methods that will be available for clients using this object. Available methods are those with simple or serializable attributes and return types.

Available: getRowStatus(Row):String

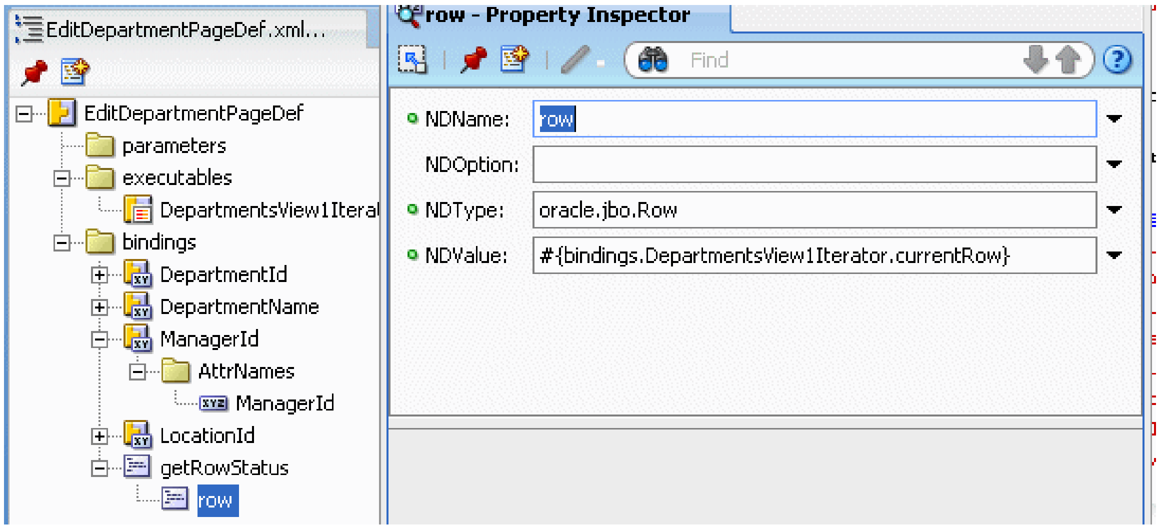
Selected: DepartmentsView

This step makes the method available in the ADF binding layer so it can be added as a method binding to the pageDef file of the edit form.

Databinding

To make the method available to the edit form, select the page definition file (e.g. EditFormPagedef.xml in the example) and choose **insert inside bindings | Generic Bindings | method Action** from the context menu of the **Bindings** node. Make sure you select "methodAction", not "Action". Select the ViewObject that exposes the *getRowStatus* method and select it.

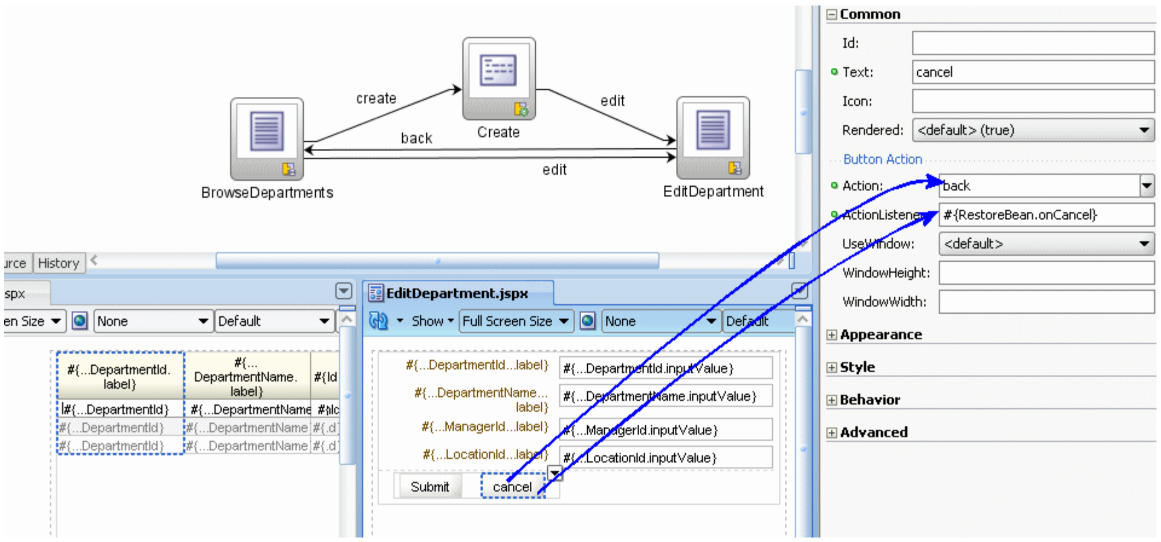
In the argument field, provide ExpressionLanguage that points to the iterator binding and there the current row. In the example, this is **#{bindings.DepartmentsView1Iterator.currentRow}**. Setting the EL ensures that the row passed to the method always is the row the users works with.



NDName:	row
NDOption:	
NDType:	oracle.jbo.Row
NDValue:	#{bindings.DepartmentsView1Iterator.currentRow}

View Layer

On the view layer there are three things to do: i) set immediate=true on the cancel button so client validation is bypassed, ii)add and reference an action listener in a managed bean to clean up the modifications and iii) navigate back to the browse page.



Common

Id:	
Text:	cancel
Icon:	
Rendered:	<default> (true)

Button Action

Action:	back
ActionListener:	#{RestoreBean.onCancel}
UseWindow:	<default>
WindowHeight:	
WindowWidth:	

Appearance

Style

Behavior

Advanced

The managed bean code access the *getRowStatus* method that is exposed on the binding to determine whether or not the row is new or an existing row. Based on the returned information, the row then is either refreshed or removed before navigation continues to the browse page

```
public void onCancel(ActionEvent actionEvent) {
  FacesContext fctx = FacesContext.getCurrentInstance();
  DCBindingContainer bindings = (DCBindingContainer)
  BindingContext.getCurrent().getCurrentBindingsEntry();
  DCIteratorBinding iter =
        bindings.findIteratorBinding("DepartmentsView1Iterator");
  Row rw = iter.getCurrentRow();

  OperationBinding getRowStatusBinding =
                 bindings.getOperationBinding("getRowStatus");
  String rwStatus = (String)getRowStatusBinding.execute();
  if ("NEW".equalsIgnoreCase(rwStatus)){
    iter.removeCurrentRow();
    iter.refreshIfNeeded();
  }
  else{
    rw.refresh(Row.REFRESH_UNDO_CHANGES);
  }
  fctx.renderResponse();
}
```

And this is it. Don't forget to set **"immediate=true"** on the cancel button.

Download Sample

The sample this how-to is based on can be downloaded 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 to access an HR schema in a local database of yours