ADF Code Corner

0007. How to cancel an edit form, undoing changes with ADFm savepoints

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
08-JAN-2009

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

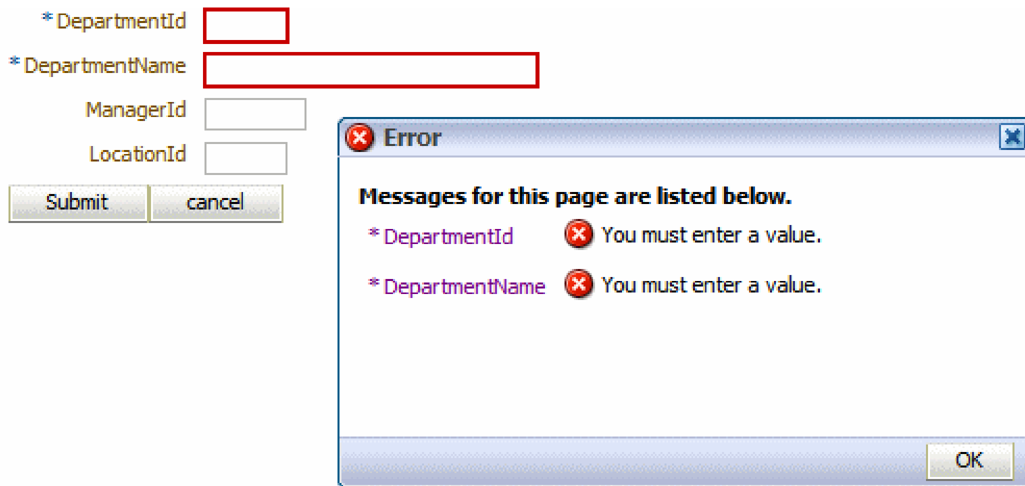
Introduction

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 save point 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. In a previous article, we introduced a Java based solution that didn't use any save point feature but a pure code centric approach.

The Java based approach however falls a bit short when a record was deleted and now needs to be recovered. In this article, the focus is on model save points that can be set programmatically or implicit in bounded taskflows.

Problem Description

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



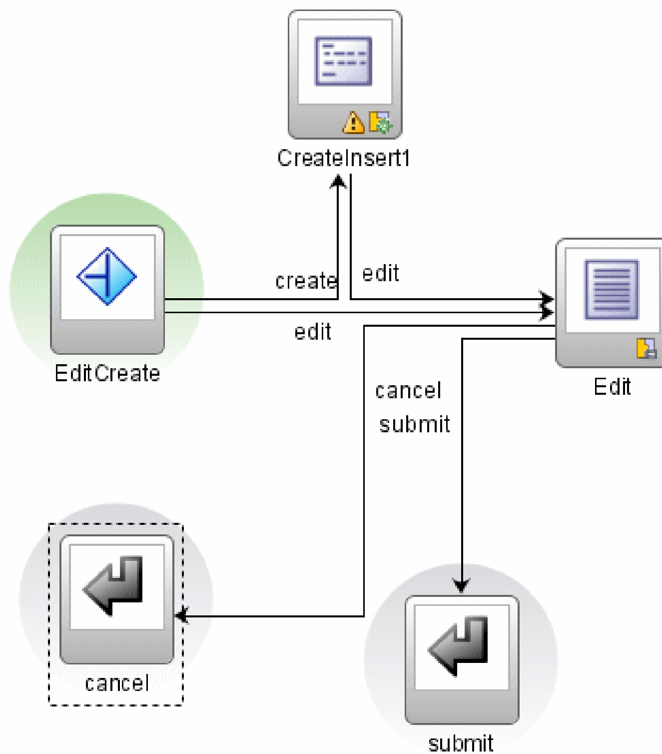
There are two issues here: i) The user didn't provided all the required field values before pressing "cancel" and ii) a new record has been created in the ADF iterator binding, which awaits clean up.

About ADFm Savepoints

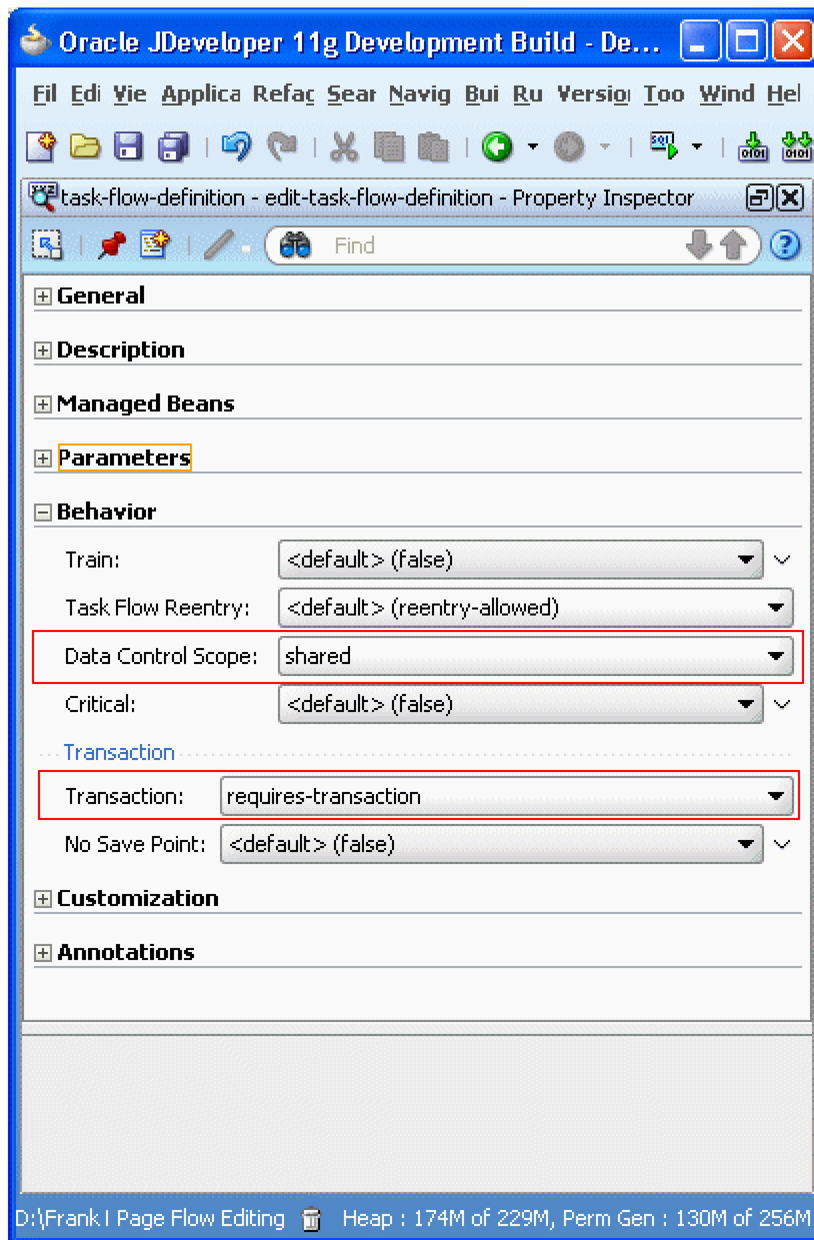
The ADF model (ADFm) save point is a snapshot of the transaction state. Restoring a model save point rolls back the current transaction up to the point when the save point was created. ADF model save points can be created explicitly by the application developer or implicit by the ADFc controller. ADFc implicitly creates a save point if a bounded taskflow is entered without starting its own transaction. The save-point is used if the user abandons the bounded taskflow, for example using the browser back button, or when a return activity has the restore-save-point element set.

Implicit ADFm Savepoints

Implicit save points are configured on bounded taskflows. As soon as a navigation enters the bounded taskflow, a snapshot of the current transaction is taken to revert to the state before entering the bounded taskflow if needed. In the example below, the bounded taskflow contains the logic for an edit form for employee data. The router determines whether it is a "new create" or an "edit" request for an existing employee record. In both cases the "Edit" view is displayed to the user to add or modify the data.



The edit form in the "Edit" view has a "submit" button and a "cancel" button. It is important to point out that the **immediate** property on the cancel button must be set to **true** to avoid field validation errors when canceling the form. The cancel button navigates the request to the "Submit" return activity, whereas the cancel button routes it to the "cancel" return activity. To enable transaction handing in a bounded taskflow, the taskflow itself must be configured. In this example, the bounded taskflow is set to use a shared **Data Control Scope** (which also is the default setting) and has its **Transaction** property set to **requires-transaction**. The **Transaction** property setting defines that the bounded taskflow either leverages an existing transaction from the calling taskflow or starts a new transaction if none exists. In his upcoming February/March column for Oracle Magazine, Steve Muench explains in detail how the declarative transaction works in bounded taskflows and you find a reference to all his Oracle Magazine articles at the end of this paper. For now, it is enough to understand that the transaction either exists or is created in the bounded taskflow.

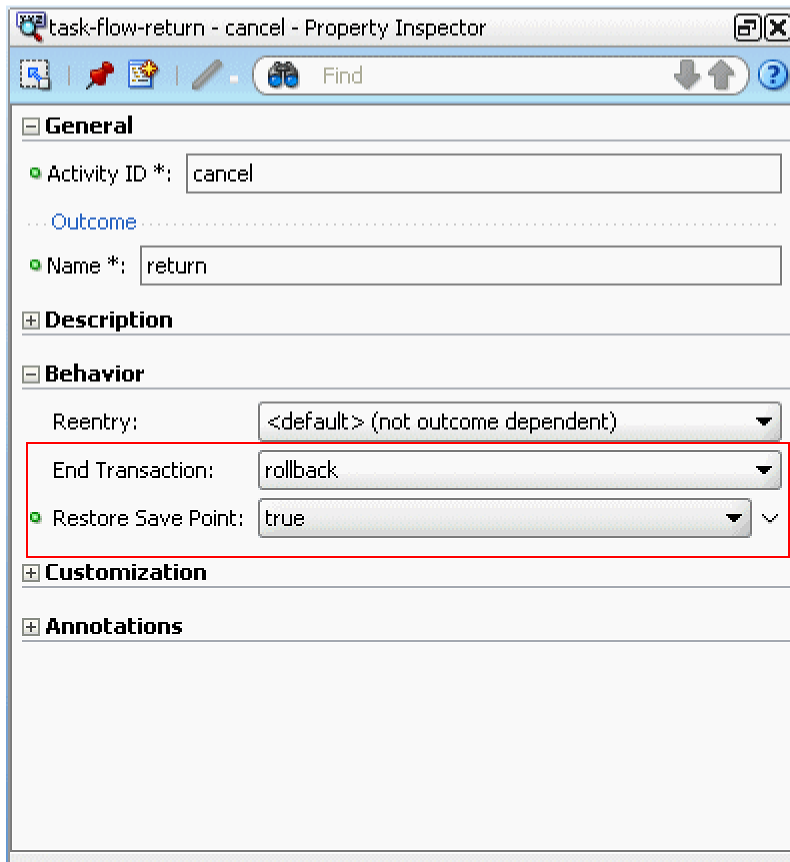


The return activities differ in their settings for the **End Transaction** and **Restore Save Point** properties. For the submit button, there is no need to restore the save point and all you need to do is to set the **End Transaction** property to **commit**. Commit ? Yes commit! Why ? Because I said so!

The reason why the **End Transaction** needs to be set to commit is that the bounded taskflow may need to create a new transaction if no existing one exists. To see the applied changes in the caller taskflow, the changes must be committed. If there exist a transaction - opened by the parent flow (or view port as it is more precise to say) - then the commit is ignored and the return happens without committing the transaction so that no damage is done to uncommitted data states.

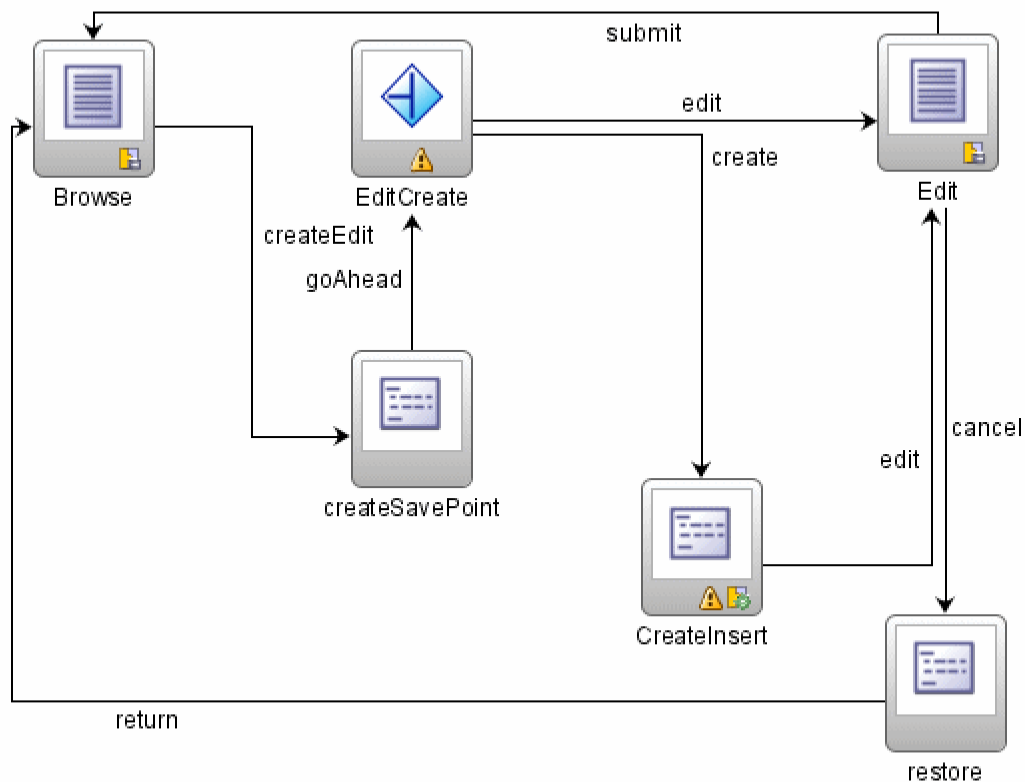
The settings for the "cancel" return activity are **rollback** for the **End Transaction** and **Restore Save Point** set to **true.** Similar, if a new transaction was created by the bounded taskflow, the rollback will

undo all the changes. If an existing transaction was used, a roll back to the previous stored save point will restore the state that existed before entering the taskflow.



Explicit ADFm Savepoints in ADF Task Flow

The explicit save point use case is in analogy to the previous article though a bit smarter in the approach. Instead of exposing a method on the Application Module, this approach programmatically sets and restores ADFm save points. The nice thing about this approach is that it not only works for bounded taskflows but unbounded taskflows as well.



The flow in the image above is similar to the flow shown with the bounded taskflow. The router activity directs the request to the "Edit" view, either for creating a new employee record or a for editing an existing record.

Again it is important to notice that the **cancel** button on the "Edit" form has its **immediate** property set to **true** to avoid filed validation errors. The ADFm save points are created and restored by method call activities that reference a method in a managed bean. Since the managed bean methods doesn't hold any state, it can be configured with a **None** scope so it doesn't hang around unused in memory. The save point methods are

```
public void restore_action() {
 String sph = (String) rctx.getPageFlowScope().get("AdfmSavePoint");
 BindingContext bctx = BindingContext.getCurrent();
 DCDataControl dcDataControl = bctx.getDefaultDataControl();
 dcDataControl.restoreSavepoint(sph);
}

public void createAdfmSavePoint(){
 BindingContext bctx = BindingContext.getCurrent();
 DCDataControl dcDataControl = bctx.getDefaultDataControl();
 String sph = (String) dcDataControl.createSavepoint();
 rctx.getPageFlowScope().put("AdfmSavePoint",sph);
}
```

The save point created by the **createAdfmSavePoint** method is stored in the taskflow's **pageFlowScope** where it is accessed from the **restore_action** method. Note that the lifetime of a save point is limited to until the transaction is committed or rolled back. In other words, you cannot commit a form to then use a save point to restore the old state. Something to keep in mind when choosing one or the other approach.

The method call activity definitions used in this example are

```
<method-call id="createSavePoint">
  <method>#{BrowseDepartmentBean.createAdfmSavePoint}</method>
  <outcome>
    <fixed-outcome>goAhead</fixed-outcome>
  </outcome>
</method-call>
<method-call id="restore">
  <method>#{BrowseDepartmentBean.restore_action}</method>
  <outcome>
    <fixed-outcome>return</fixed-outcome>
  </outcome>
</method-call>
```

Download Samples

The two sample workspaces are build against the HR demo schema using Oracle JDeveloper 11 production. You can download the sample workspacess from the ADF Code Corner website:
http://www.oracle.com/technetwork/developer-tools/adf/learnmore/index-101235.html

RELATED DOCOMENTATION

☒	Steve Muench's Oracle Magazine columns - http://www.oracle.com/technology/products/jdev/tips/muench/oramag/index.html
☒	Adding savepoints to a Task Flow - Adding savepoints to a Task Flow - http://download.oracle.com/docs/cd/E15523_01/web.1111/b31974/taskflows_complex.htm#CHDBDJBI
☒	