# ADF Code Corner

## 94. ADF Region Return Value Strategy


twitter.com/adfcodecorner

**Abstract:**

While bounded task flows can return values upon exit if called from a task flow call activity, its not like this when calling task flows from an ADF region. One reason why return values don't work for regions is because regions are not expected to be exited. Another reason is that the task flow binding that exposes the bounded task flow as a `RegionModel` to the `RichRegion` UI component does not have a returnValue property.

This article shows how to return values from ADF regions upon region task flow exit. The implementation uses a shared context bean passed to the bounded task flows as an input parameter, similar to how the Dynamic Tab Shell template uses it.

Author: Frank Nimphius, Oracle Corporation
twitter.com/fnimphiu
12-JAN-2012

*Oracle ADF Code Corner is a loose blog-style series of how-to documents that provide solutions to real world coding problems.*

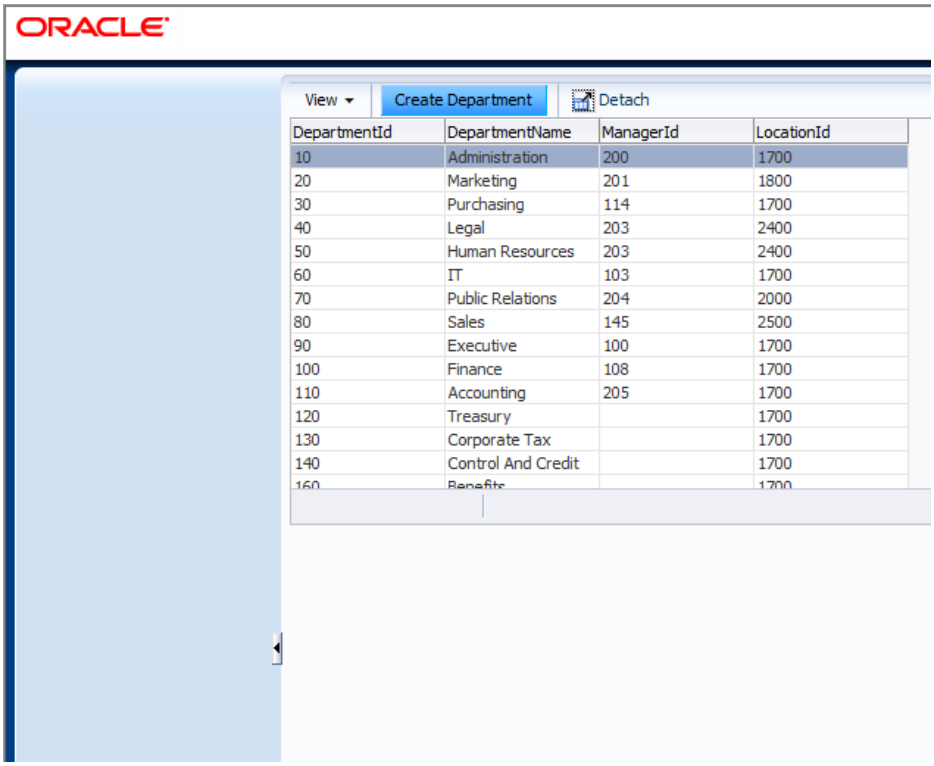
*Disclaimer: All samples are provided as is with no guarantee for future upgrades or error correction. No support can be given through Oracle customer support.*

*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*

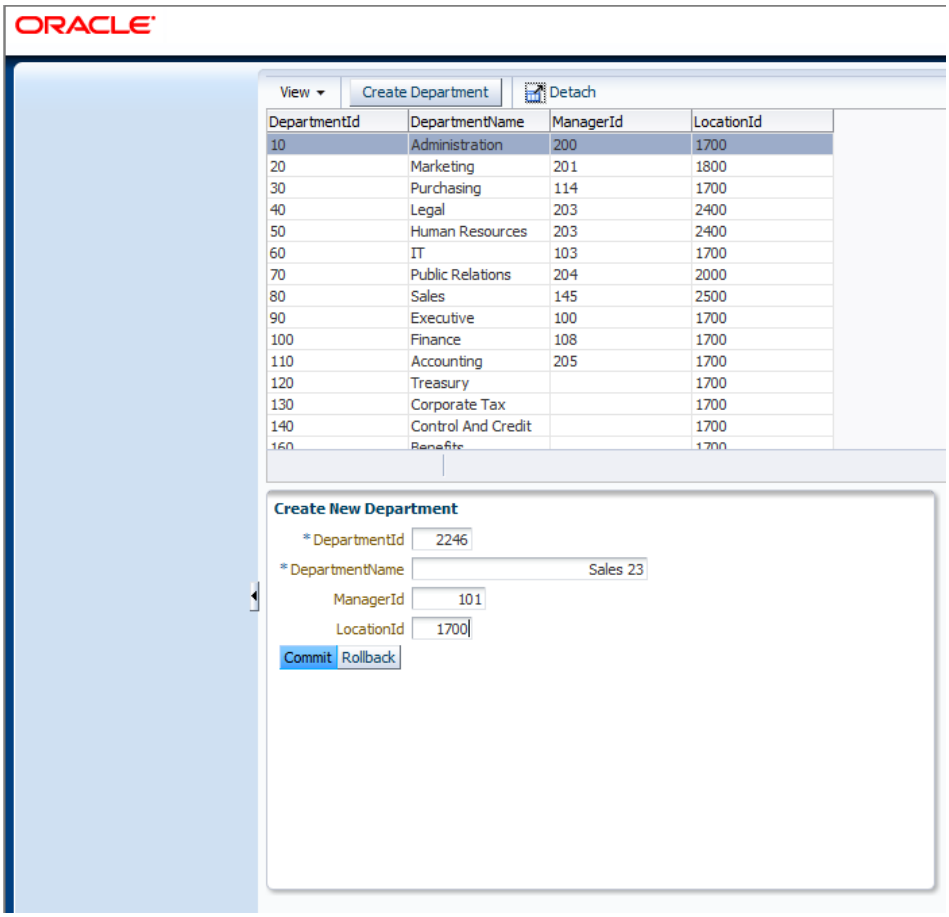
## Introduction

ADF regions don't have a **returnValues** property defined for values to be returned upon region exit. However, only because there is no property to return a value doesn't mean it is not possible.

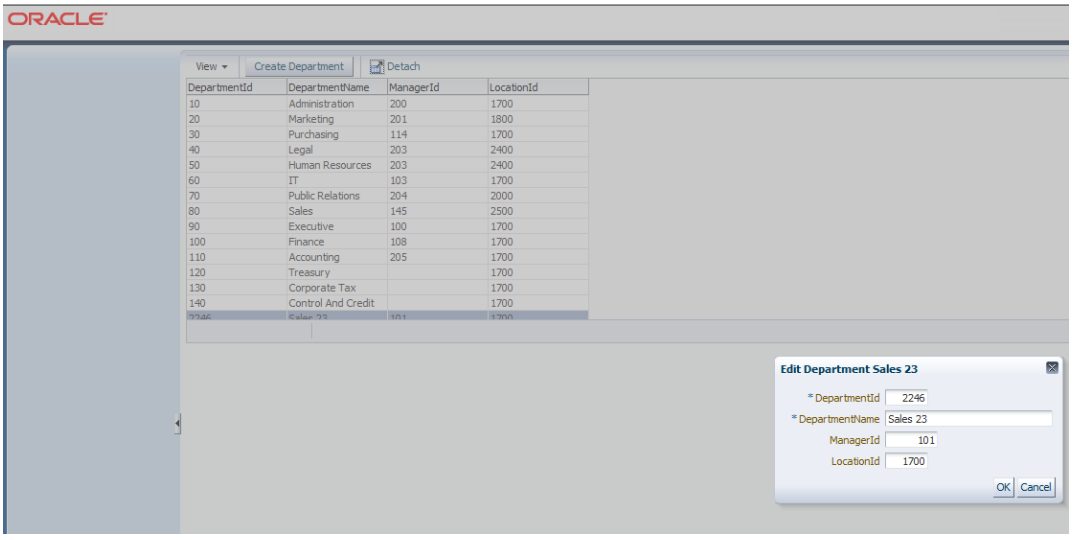
The sample below allows users to create a new department by pressing the **Create Department** button. The department – for demonstration purpose – is created by a bounded task flow exposed as a region.




The bounded task flow is configured to not share the Data control (which makes sense for the use case as otherwise the Data Control could be used as a data exchange layer).




The user either commits the new department or cancels the action. In both cases, the bounded task flow navigates to a return activity that exists the bounded task flow. So the first lesson to learn from this is that task flows in regions can be exited (though you have to handle the exit case as shown later).

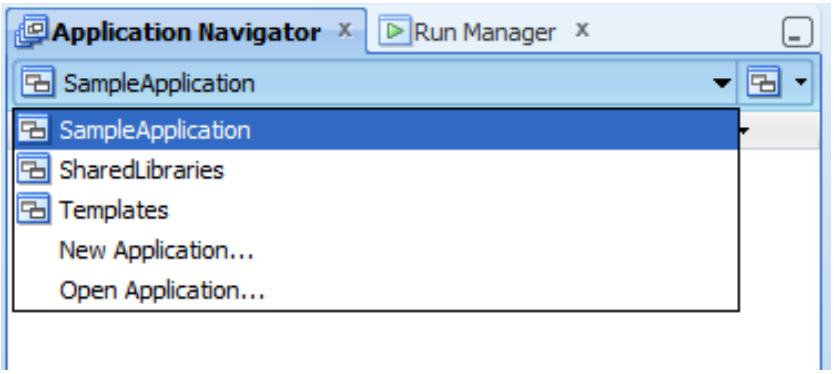


Upon task flow exist, the parent view looks at whether the exit happened on commit or roll back. If it happened on commit, then the departments table iterator is refreshed and the new department opened in a popup window. For this, two information need to be passed as return values from the bounded task flow to the parent view, for which I used a shared context bean.

## Application Architecture

The application is a bit more complex than usual on ADF Code Corner. Three workspaces are used for this solution. The **SampleApplication** workspace contains the Oracle ADF BC model, the bounded task flow project and the ViewController project. The **SharedLibraries** workspace contains the two classes I use to create and pass a shared context bean to the bounded task flow in the region.

The **Templates** workspace contains the bounded task flow template for this implementation. The thinking behind all these applications is reuse and best practices (though not designed to max as architecture is not the focus of this article). Using a context bean can be looked at as a a strategy or convention. I don't consider it a pattern because it makes assumption about custom APIs to be available.




## Shared Context

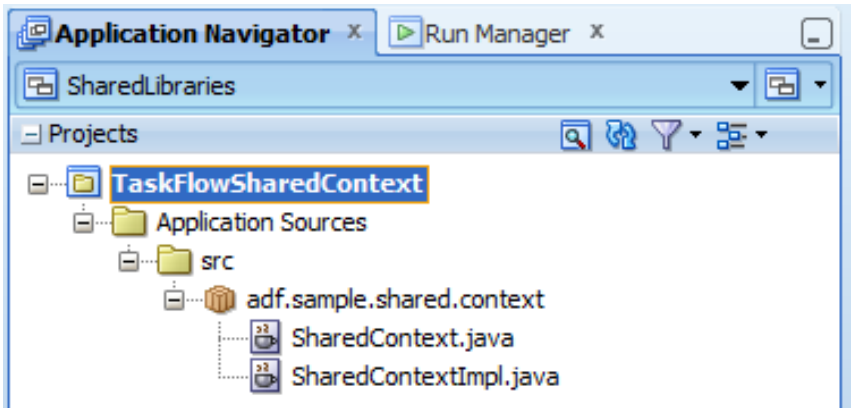
The shared context is a managed bean instance that the parent view, the view with the departments table, creates and then provides as an input parameter to the bounded task flow in the region.

The **SharedLibraries** JDeveloper workspace defines two Java classes, `SharedContext.java` and `SharedContextImpl.java`.

The `SharedContext.java` class is the input parameter type of the bounded task flow that holds the shared bean class. The `SharedContextImpl.java` class extends `java.util.HashMap` and is the object shared between the parent flow and the called flow (region).

Why extending `HashMap`? Ideally you work with bean properties as the interface so you have a type save programming interface with the ability to define a default value and to document its use. However, it may

happen that there is a single use case that requires one more parameter. This then could be easily added by a call to put() on the extended `HashMap` instance, or by using #{bean['key']} in EL. So the HashMap is a bit of precaution that helps with unpredictable requirements that may not justify a change in the API.




**SharedContext**

```
/**
 * Managed bean to reference the shared context object
 */
public class SharedContext {
  SharedContextImpl currentInstance = null;
  public SharedContext() {
      super();
  }
  public void setCurrentInstance(SharedContextImpl currentInstance) {
      this.currentInstance = currentInstance;
  }
  public SharedContextImpl getCurrentInstance() {
        return currentInstance;
  }
}
```

**SharedContextImpl**

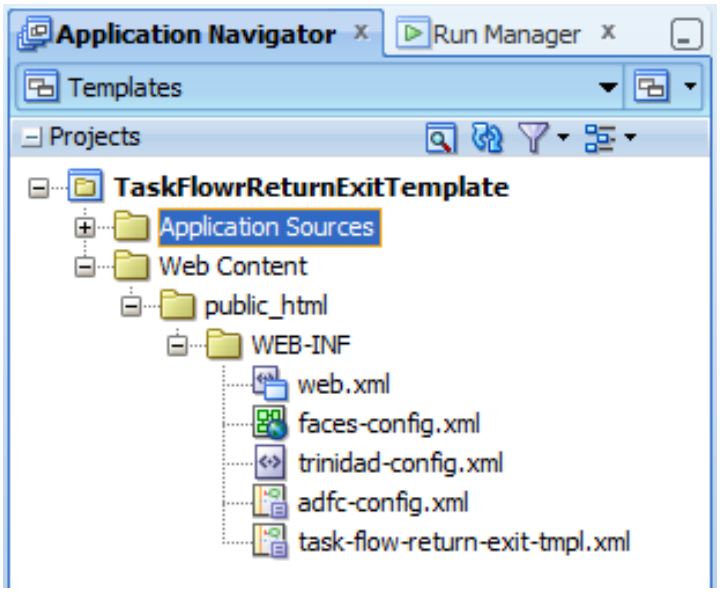
```
import java.util.HashMap;
/**
 * Shared Context class extending Hashmap. All shared information is
 * exposedthrough public methods but saved in the HashMap. Developers
```

```
 * are able to pass addition information to a task flow by directly
 * calling put / get on the context class. */
public class SharedContextImpl extends HashMap{
  @SuppressWarnings("compatibility")
  private static final long serialVersionUID = -2047438954870030787L;

  final String RETURN_VALUE = "___returnValue_____";
  final String COMMIT_INDICATOR = "___commitIndicator_____";

  Boolean exitIsCommit = false;

  public SharedContextImpl() {
      super();
      this.put(RETURN_VALUE, null);
      this.put(COMMIT_INDICATOR, exitIsCommit);
  }

  public void setExitIsCommit(Boolean exitIsCommit) {
     this.put(COMMIT_INDICATOR, exitIsCommit);
  }


  public Boolean getExitIsCommit() {
     return (Boolean) this.get(COMMIT_INDICATOR);
  }

  //define some helper classes used in a task flow
  //method call activity
  public void setReturnValue(Object returnValue) {
      this.put(RETURN_VALUE, returnValue);
  }

  public Object getReturnValue() {
    return this.get(RETURN_VALUE);
  }

  public void setCommittedonExitTrue(){
    this.put(COMMIT_INDICATOR, Boolean.TRUE);
  }

  public void setCommittedonExitFalse(){
     this.put(COMMIT_INDICATOR, Boolean.FALSE);
  }
}
```

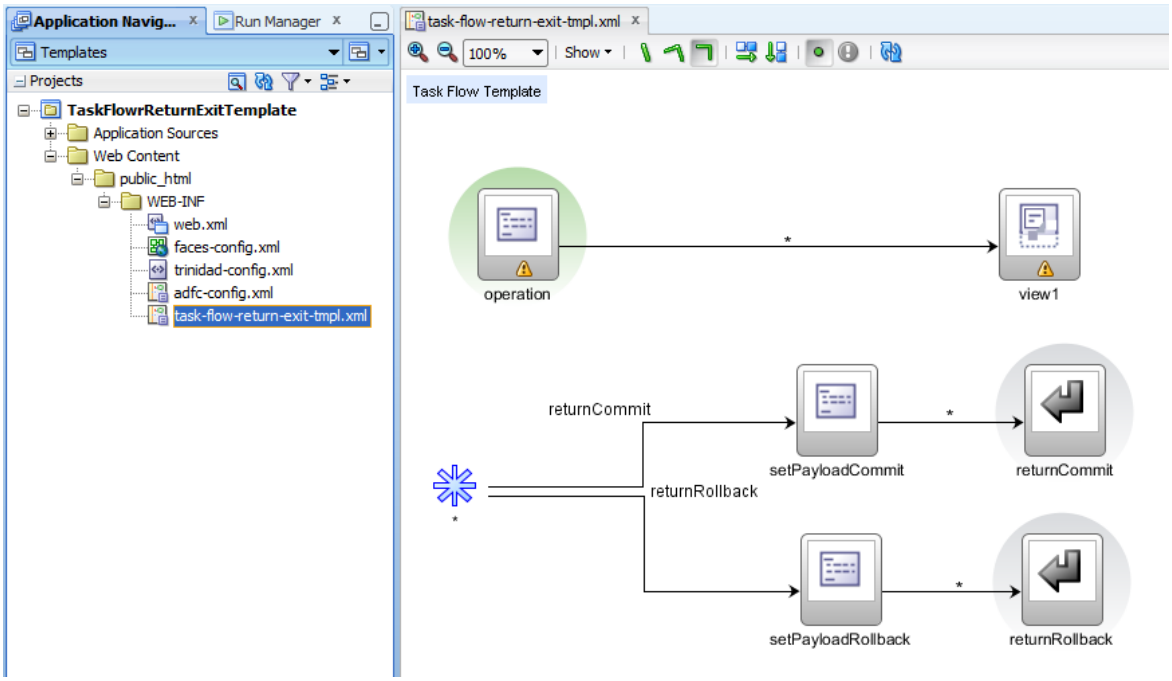
The shared context files are deployed in a client JAR file so they can be added as a library to the bounded task flow and ViewController project.

## Task Flow Template

The task flow template defines the contract between the parent flow and the sub flow. Building a new bounded task flow from this template (using the copy option!) creates a basic flow and the context bean input parameter definition.




The task flow template defines a method call activity as the default activity. This method call activity will create the new row for a collection. When developing the bounded task flow, you simply drag a method or operation from the Data Controls panel onto the call activity. The view activity then is used to define the user interface for creating a new department. If course, the bounded task flow could have more than one view if needed. This sample however is okay with one view.



The key concept for the strategy used in the sample is the return activities, one for the commit case and one for the rollback case.

The return activities are accessible from a wild card flow so any activity in a bounded task flow can access them. The method call activities reference the `SharedContextImpl` class instance where they call one of the helper classes, to indicate how the task flow (region) was exited.

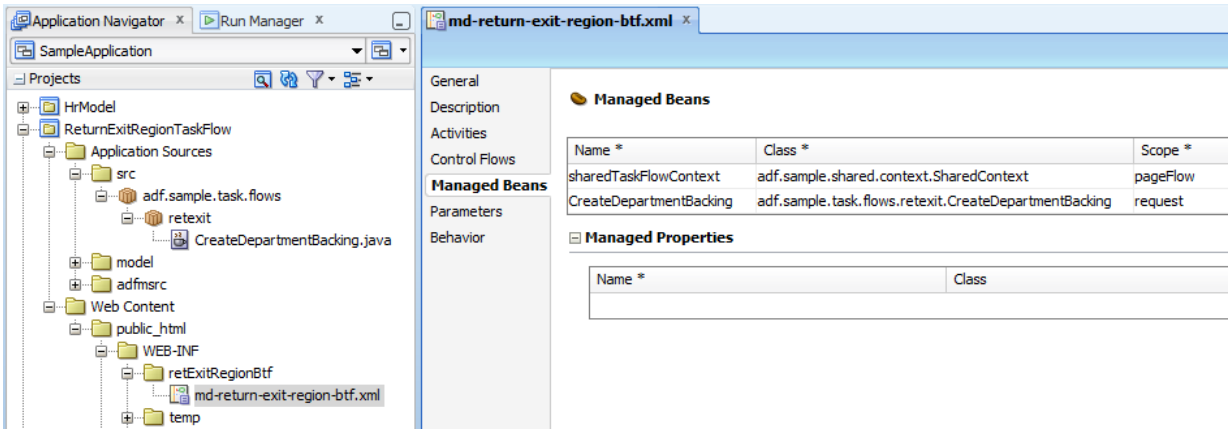
The template then is deployed as an ADF library so it can be reused in other ADF applications (which makes sense from a best practices point of view).

## Bounded Task Flow (Region)

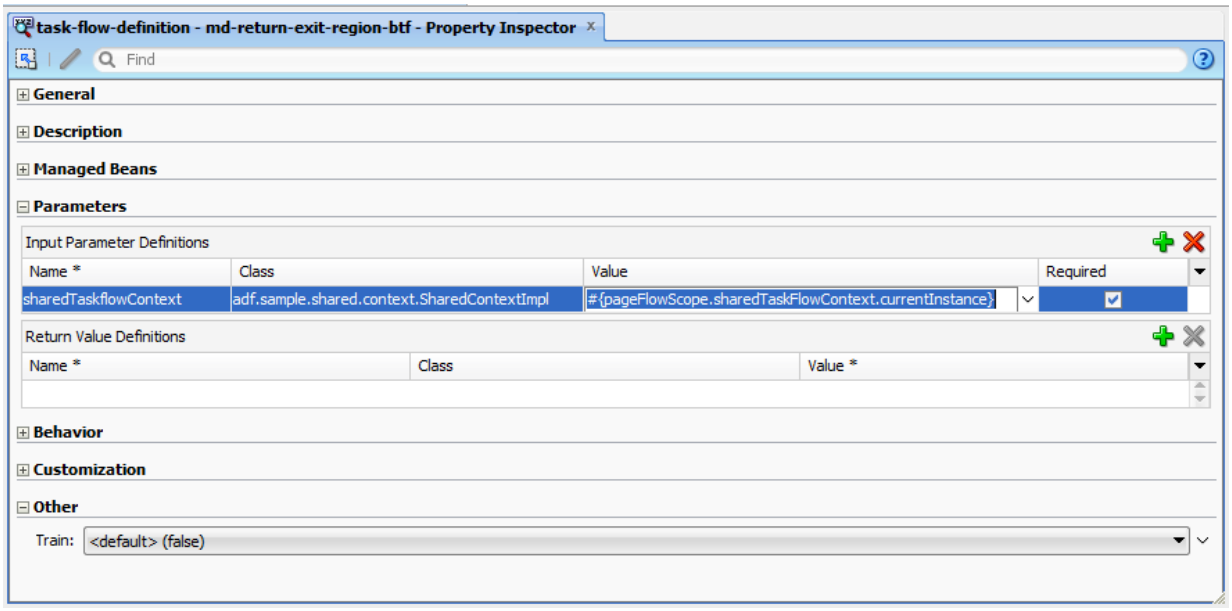
The bounded task flow gets most of its configuration from the task floe template, which it copies upon task flow creation (it's a dialog option). There are two managed beans defined in the bounded task flow for this sample. The **sharedTaskFlowContext** managed bean is defined by the template.

The **CreateDepartmentsBacking** bean is used to add the departmet Id of the new department to the shared context so it is returned upon task flow exit. The bean is used only when the commit button is pressed.

**Note**: For some reason I need to investigate, simply using an `af:setPropertyListener` did not work for me. Maybe the need of me using a managed is a temporary work around only. The code it uses to read the department Id and write it to a bean in pageFlow scope may become useful for you to have anyway.




The task flow input parameter that received the shared context bean bean is configured as shown below. There is no need for you to do anything to make it work (it is all defined by the template).




Notice the value reference **#{pageFlowScope.sharedTaskFlowContext.currentInstance}** that points to the managed bean defined in the bounded task flow. The managed bean is of type SharedContext and exposes a property **currentInstance** that is used to set/read the `SharedContextImpl` instance.

**CreateDepartmentBacking**

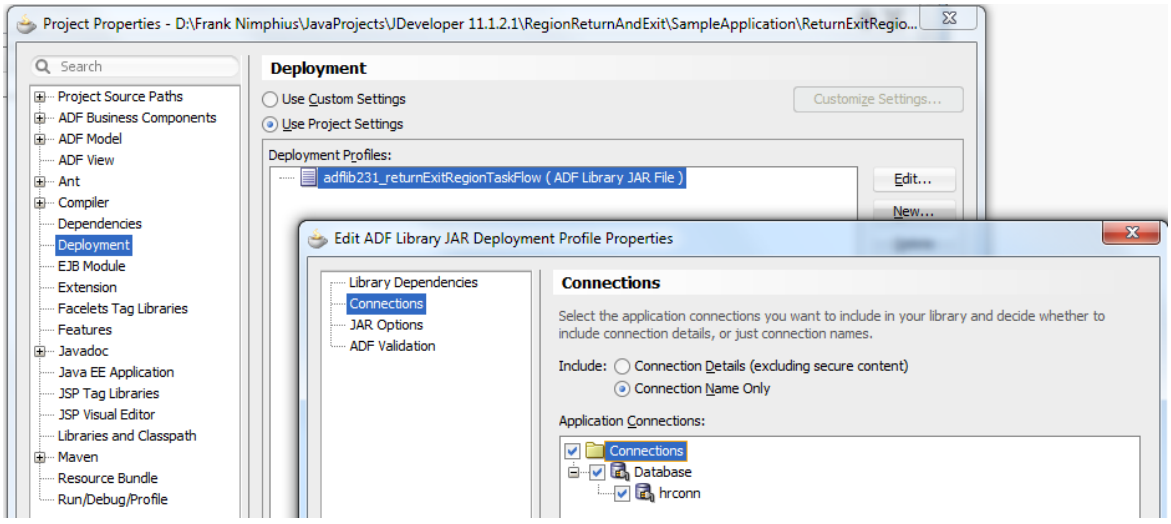
```
import adf.sample.shared.context.SharedContext;
import java.util.Map;
import oracle.adf.model.BindingContext;
import oracle.adf.share.ADFContext;
import oracle.binding.AttributeBinding;
import oracle.binding.BindingContainer;

public class CreateDepartmentBacking {
  public CreateDepartmentBacking() {
  }

  //When the commit button is pressed, read the new roe's department Id
  //value and return it to the calling task flow (parent flow)
  public String onCommit() {
    ADFContext adfctx = ADFContext.getCurrent();
    Map<String, Object> flowScope = adfctx.getPageFlowScope();
    /* its safe to assume that the managed bean is intantiated and
     * that it exists in memory because it is referenced from the task
     * flow input parameters upon task flow start. The managed bean is
     * defined in the task flow definition
     *
     * Note that if you cannot be sure a managed bean exists in memory,
     * it is better to use EL to access it so that JSF can create an
     * instance of the bean, avoiding a NPE

     */
```

```
    SharedContext sharedSampleContext =
       (SharedContext)flowScope.get("sharedTaskFlowContext");
    BindingContext bctx = BindingContext.getCurrent();
    BindingContainer bindings = bctx.getCurrentBindingsEntry();
    AttributeBinding deptId =
              (AttributeBinding) bindings.get("DepartmentId");

    //Save department Id value as return value
    sharedSampleContext.getCurrentInstance().setReturnValue
                                    (deptId.getInputValue());

    return "returnCommit";
   }
}
```

The bounded task flow is also deployed as an ADF library. The database connection name is deployed with the ADF library, not the credentials.

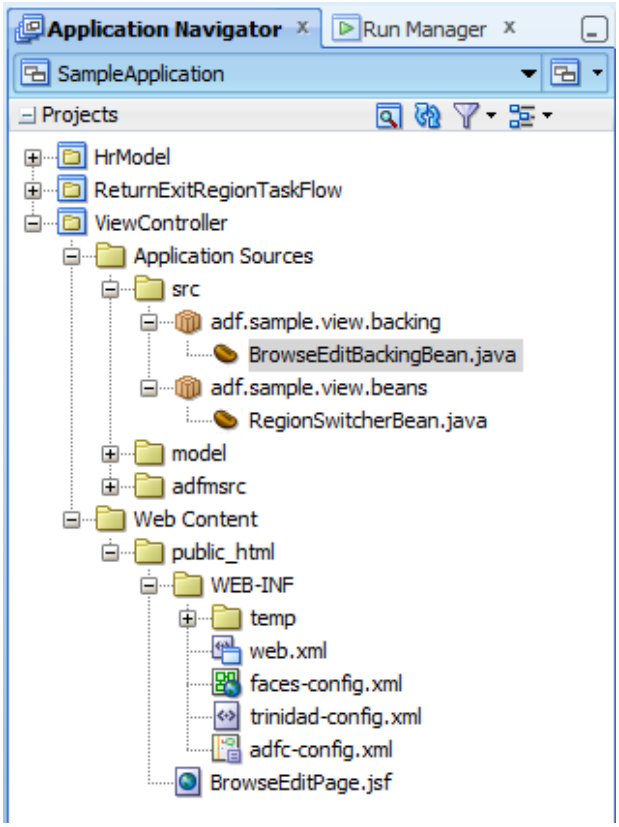
The sample application also has the database connection defined as **hrconn** so that there is no addition work required to consume the bounded task flow in the parent project. Probably also a good practice to think about common database connect names.

**Note:** The bounded task flow is created within the **SampleApplication** workspace, which – if I had to write the sample again – I would not do again. Instead I would use a separate workspace for the bounded task flow projects as well, so that the **SampleApplication** workspace only contains the application related projects (I would also move the model project into its own workspace and use it from an ADF library. For this sample however, I did not want to go too complex)




## Sample Application

The sample application's **HrModel** project is quite simple and therefore not worth discussing here. Its built by running the ADF Business Components from table wizard only.




The **ViewController** project has two Java classes defined: The **BrowseEditBackingBean** is configured as a backing bean to provide access to the popup and table component. Using a JSF component binding to a managed bean is the easiest way of getting component access at runtime.

```
package adf.sample.view.backing;

import oracle.adf.view.rich.component.rich.RichPopup;
import oracle.adf.view.rich.component.rich.data.RichTable;

 public class BrowseEditBackingBean {
   private RichTable departmentsTable;
   private RichPopup editDialog;
   public BrowseEditBackingBean() {
 }

 public void setDepartmentsTable(RichTable departmentsTable) {
   this.departmentsTable = departmentsTable;
 }


 public RichTable getDepartmentsTable() {
   return departmentsTable;
 }

 public void setEditDialog(RichPopup editDialog) {
   this.editDialog = editDialog;
```

```
 }

 public RichPopup getEditDialog() {
   return editDialog;
 }
}
```

The **RegionSwitcherBean** is a managed bean defined in **viewScope**. The bean has two responsibilities:

- It is referenced by the task flow region binding for the application to switch between two views (dynamic region). The first view is an empty task flow, which you don't need to explicitly define but show by returning an empty ID for the region.
- It defines a region navigation listener that is called whenever navigation occurs within a region. The listener in the sample waits for **null** to become the current viewId, which then indicates that the task flow has been exited. If this is the case, the listener shows the empty region and then refreshes the departments table if the exit was **onCommit**.

```
import adf.sample.shared.context.SharedContextImpl;
import adf.sample.view.backing.BrowseEditBackingBean;

import javax.el.ELContext;
import javax.el.ExpressionFactory;
import javax.el.ValueExpression;


import javax.faces.context.FacesContext;


import oracle.adf.model.BindingContext;

import oracle.adf.model.binding.DCIteratorBinding;

import oracle.adf.view.rich.component.rich.RichPopup;

import oracle.adf.view.rich.context.AdfFacesContext;

import oracle.adf.view.rich.event.RegionNavigationEvent;

import oracle.binding.BindingContainer;
import oracle.jbo.Key;
import oracle.jbo.Row;

public class RegionSwitcherBean {
  private String taskFlowId =
                "/WEB-INF/retExitRegionBtf/md-return-"+
                "exit-region-btf.xml#md-return-exit-region-btf";
  //an empty task flow is represented by an empty string
  private String emptyTaskFlow = "";
  private String currentTaskFlow = emptyTaskFlow;
```

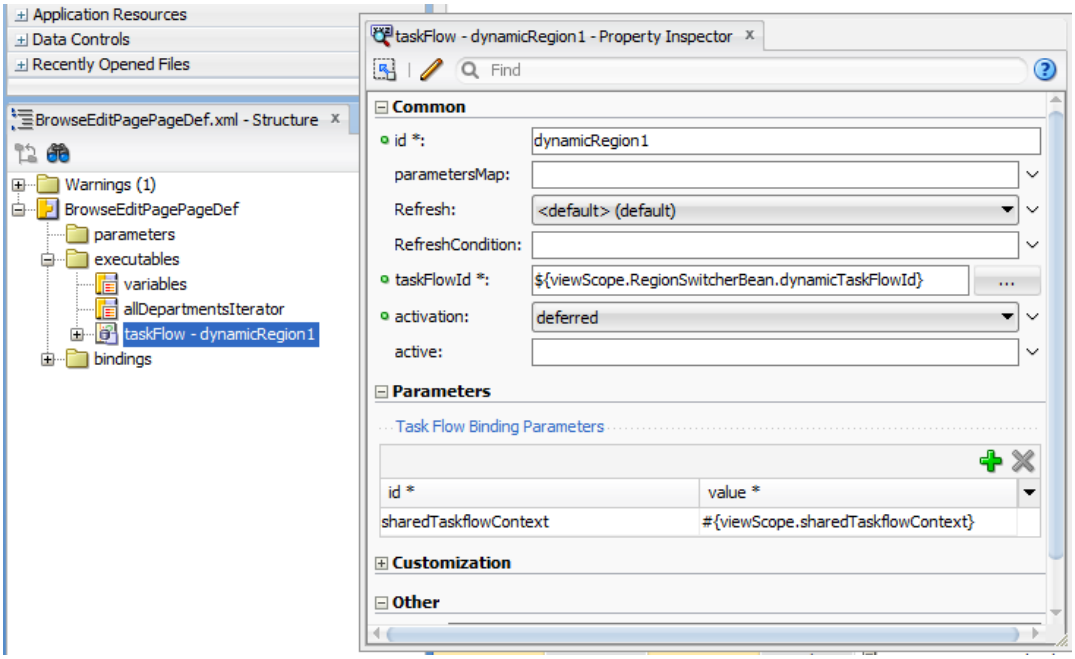
```
public RegionSwitcherBean() {}

//referenced by dynamic region
public String getDynamicTaskFlowId() {
   return currentTaskFlow;
}

//method to switch region to empty task flow
public String switchTaskFlowToEmptyString(){
 currentTaskFlow = emptyTaskFlow;
 return null;
}


//method to switch task flow to create department region. This
//method is called from the CreateDepartment buton
public String switchTaskFlowToCreateDepartment(){
   currentTaskFlow = taskFlowId;
   return null;
}
 public void onRegionNavigation(RegionNavigationEvent
                                regionNavigationEvent) {
  //check if task flow as region is exited, in which
  //case the view id is null
  if(regionNavigationEvent.getNewViewId() == null){
       currentTaskFlow = emptyTaskFlow;
       //check if exit was on commit or rollback
        FacesContext fctx = FacesContext.getCurrentInstance();
        ELContext elctx = fctx.getELContext();
        ExpressionFactory exprFactory =
                 fctx.getApplication().getExpressionFactory();
        ValueExpression ve = exprFactory.createValueExpression(
                             elctx,
                             "#{viewScope.sharedTaskflowContext}",
                             Object.class);
        SharedContextImpl sharedCtx =
                           (SharedContextImpl) ve.getValue(elctx);


       if(sharedCtx != null){
         boolean commit = sharedCtx.getExitIsCommit();
         if(commit==true){
           //set new row to become current and refresh table
         BindingContainer bindings =
              BindingContext.getCurrent().getCurrentBindingsEntry();
         DCIteratorBinding dciter =
         (DCIteratorBinding) bindings.get("allDepartmentsIterator");
```

```
            dciter.executeQuery();
            //get the return value
            Object retValue = sharedCtx.getReturnValue();
            if(retValue != null){
               //make new row current
               Key k = new Key(new Object[]{(Integer) retValue});
               Row rw = dciter.getRowSetIterator().getRow(k);
               dciter.getRowSetIterator().setCurrentRow(rw);

              //lookup table component instance using managed bean
              //reference

              ve = exprFactory.createValueExpression(
                    elctx, "#{BrowseEditBackingBean}", Object.class);
              BrowseEditBackingBean browseEditBackingBean =
                           (BrowseEditBackingBean) ve.getValue(elctx);

             //refresh table
              AdfFacesContext adfFacesCtx =
                                 AdfFacesContext.getCurrentInstance();
             adfFacesCtx.addPartialTarget(
                            browseEditBackingBean.getDepartmentsTable());

             //launch dialog to edit newly created record
             RichPopup popup = browseEditBackingBean.getEditDialog();
             RichPopup.PopupHints hints = new RichPopup.PopupHints();
             popup.show(hints);
           }
           else{
             //Please use AdfLogger in your application development
             System.out.println("LOG: No return value found");
           }
         }
         else{
           //nothing to do
         }
       }
       else{
        System.out.println("LOG: Shared Context is NULL");
       }
     }
   }
}
```

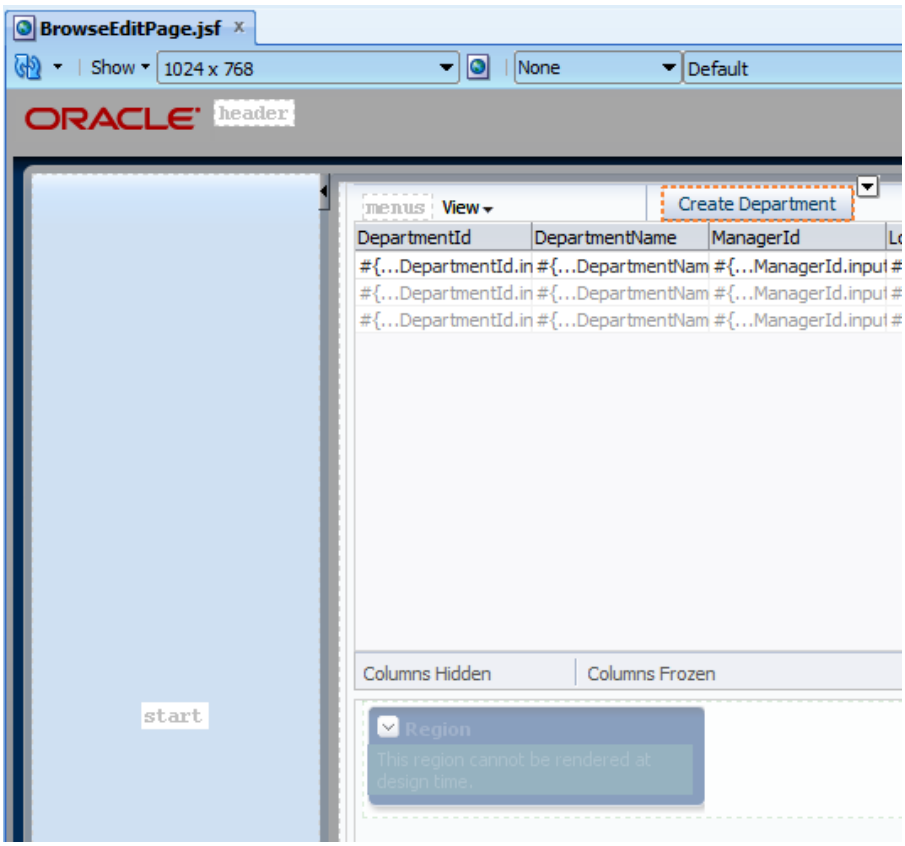
When the bounded task flow is added to the parent view, a task flow binding is created in the **executables** section of the **PageDef** file.

The task flow binding references the **RegionSwitcherBean** bean to obtain the task flow id of the task flow to show in the region. The task flow binding input parameter **sharedtaskFlowContext** is created

because it is a require input parameter of the bounded task flow. The parameter references a managed bean of type `SharedTaskFlowContext` which is created in the parent view ADF controller configuration (adfc-config.xml in the sample).




```
<?xml version="1.0" encoding="windows-1252" ?>
<adfc-config xmlns="http://xmlns.oracle.com/adf/controller" version="1.2">
  <managed-bean id="__1">
    <managed-bean-name>RegionSwitcherBean</managed-bean-name>
    <managed-bean-class>adf.sample.view.beans.RegionSwitcherBean</managed-bean-class>
    <managed-bean-scope>view</managed-bean-scope>
  </managed-bean>
<managed-bean id="__2">
    <managed-bean-name>sharedTaskflowContext</managed-bean-name>
    <managed-bean-class>adf.sample.shared.context.SharedContextImpl</managed-bean-class>
    <managed-bean-scope>view</managed-bean-scope>
  </managed-bean>
  <managed-bean id="__3">
    <managed-bean-name>BrowseEditBackingBean</managed-bean-name>
    <managed-bean-class>adf.sample.view.backing.BrowseEditBackingBean</managed-bean-class>
    <managed-bean-scope>request</managed-bean-scope>
  </managed-bean>
</adfc-config>
```

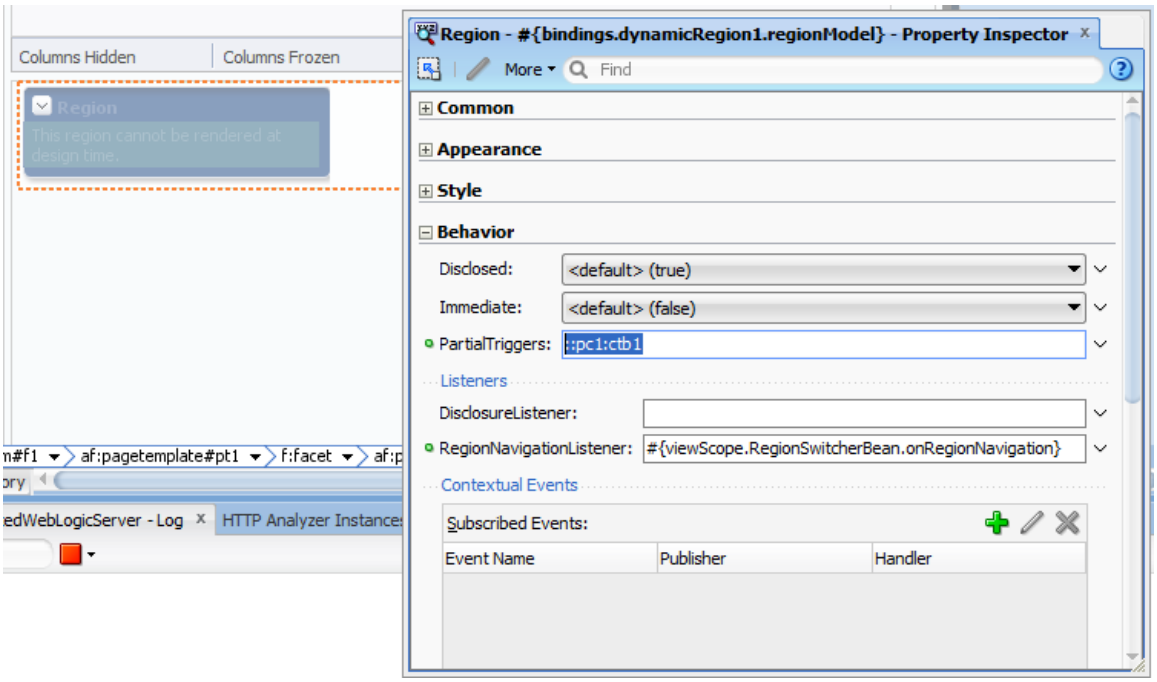



The Create Department command switches the region view to display the bounded task flow and also is referenced by the `af:region` **PartialTriggers** property so the region refreshes. A good enhancement for this sample would be to disable the command button when the create department bounded task flow shows in the region and to enable it when the task flow exits. I leave this as a homework for you ;-)

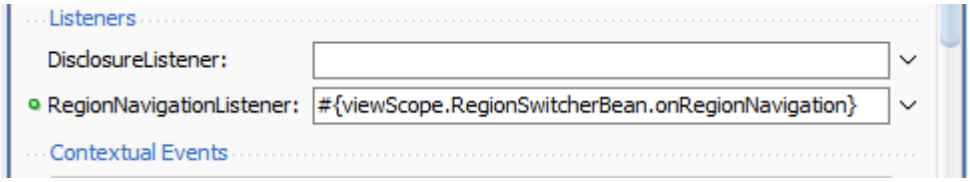
The code to switch the empty task flow view to the create department task flow is shown below

```
//method to switch task flow to create department region
public String switchTaskFlowToCreateDepartment(){
    currentTaskFlow = taskFlowId;
    return null;
}
```

Shown in the image below is the **PartialTriggers** property reference of the ADF region to the command button.




The **RegionNavigationListener** also is defined on the `af:region` tag and points to the RegionSwictherBean.




## Summary

Though AD regions don't have a return value property, it can be implemented with a design strategy using a shared context bean. The sample explained in this article can be customized to also suit other use cases, for example to pass information back to the calling task flow without exiting the bounded task flow. All you need is a flag that indicates to the **RegionListener** that information should be passed back to the parent flow.

**Downloading the Sample**

The sample application is built with JDeveloper 11.1.2.1 and can be downloaded as sample 94 from the ADF Code Corner website.

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

The database connection of the **SampleApplication** workspace needs to be changed to point to the HR schema of your database instance.

Run the **BrowseEditPage.jsf** page in the **ViewController** project to try the sample.

**Note:** Though the sample is written for JDeveloper 11.1.2.1, the strategy followed can be used with JDeveloper 11g R1 versions as well.

## Known Issues:

Not sure if it is my setup, but the popup did not come up on FireFox 3.6 and instead the page was refreshed. Upgrading FireFox solved this problem for me. Internet Explorer and Chrome did fine.

**RELATED DOCOMENTATION**

| | |
|---|---|
| ⊠ | |
| ⊠ | |
| ⊠ | |