ADF Code Corner

97. How-to defer train-stop navigation for custom form validation or other developer interaction

Abstract:

ADF developers can declaratively define a bounded task fow to expose a train model for users to navigate between views. As power comes with complexity, there is a lot you can do with trains if you go beyond drag-and-drop in ADF. This article explains how user train stop selections can be intercepted, followed or suppressed by the developer. The sample provided with this article displays a popup dialog for the user to confirms he/she wants to navigate off the current view. If the user presses cancel, no navigation is performed.

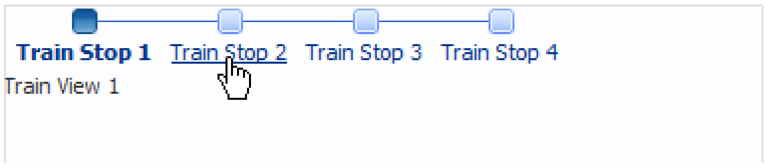
twitter.com/adfcodecorner

Author: Frank Nimphius, Oracle Corporation
twitter.com/fnimphiu
21-FEB-2011

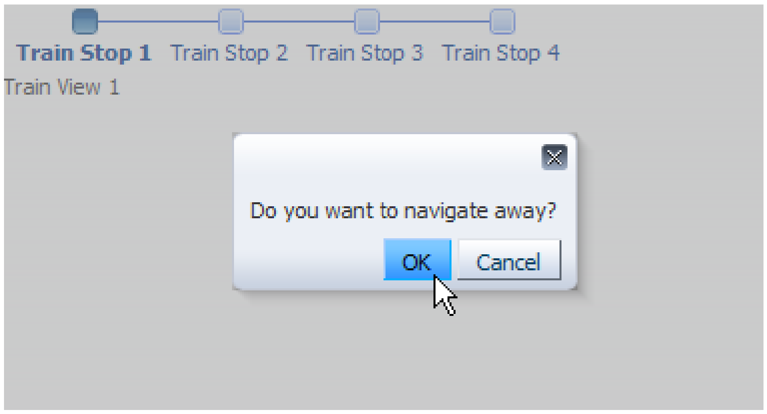
Introduction

The train component and the associated train model in bounded task flows is a powerful feature in ADF that can be tailored and customized for different kind of cases beyond simple view to view navigation. In this article, the user train stop selection to naigate to another view is intercepted for the developer to object and cancel or continue with the navigation.

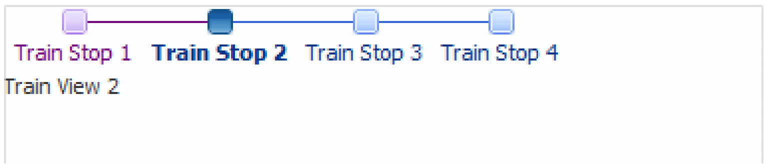
The screenshots below are taking from the sample built for this code corner article. In this sample, when a user clicks on a train stop icon to perform navigation …



… instead of navigating to the view associated with the train stop, a popup is displayed. The popup, for example, could be shown if custom validation fails for a view or if data needs to be be committed.



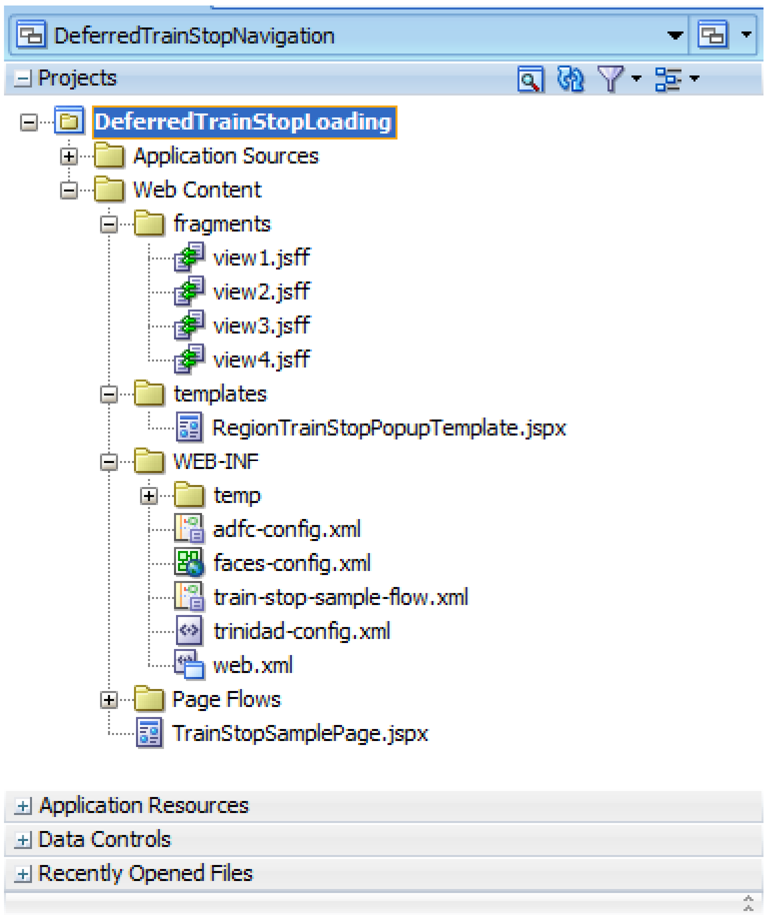
If the user **OK**s the dialog, navigation is performed according to the train navigation rule. If the user cancels the navigation, the current view is not left.



Implementing the solution

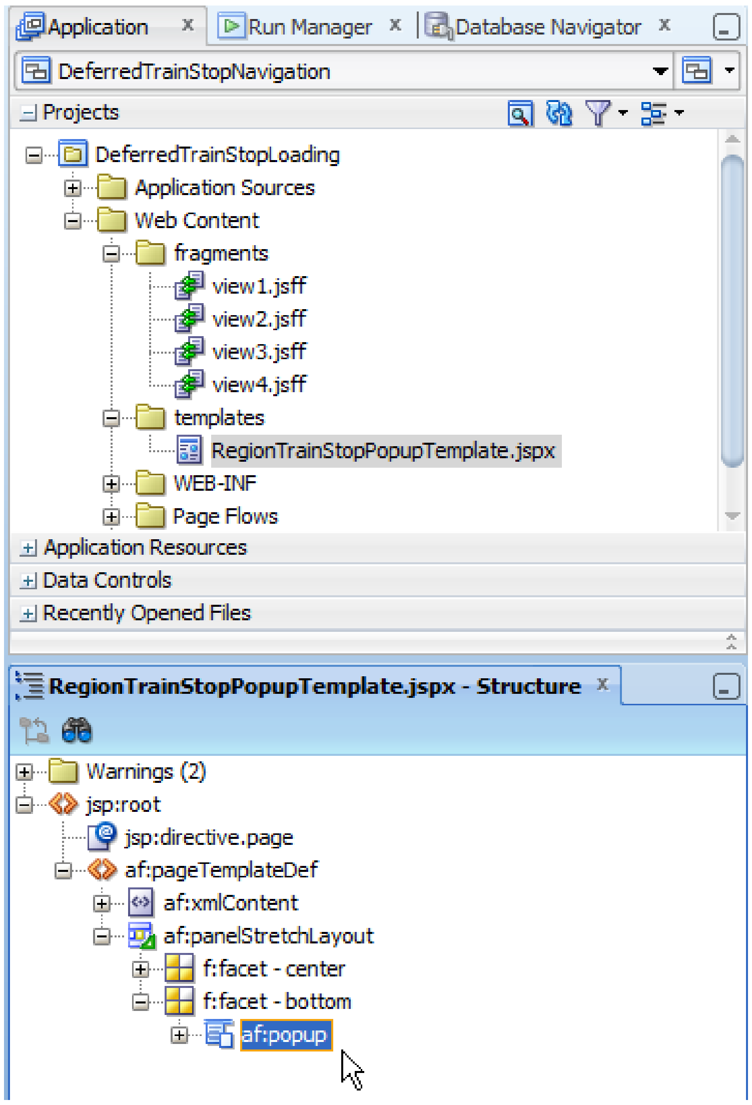
Implicit named navigation cases are used in ADF when navigating between views in a bounded task flow using the train model. To intercept the navigation request, which is triggered by the user clicking onto a train stop in an **af:train** component or on a train button in a **af:trainButtonBar** component, you need to customize the train **nodeStamp** facet, adding a custom command item that routes all train stop actions to a managed bean first.

But, let's go step-by-step. The sample has four views displayed in a train. Each view is based on a page template that contains a popup component.



```
DeferredTrainStopNavigation
Projects
DeferredTrainStopLoading
    Application Sources
    Web Content
        fragments
            view1.jsff
            view2.jsff
            view3.jsff
            view4.jsff
        templates
            RegionTrainStopPopupTemplate.jspx
        WEB-INF
            temp
            adfc-config.xml
            faces-config.xml
            train-stop-sample-flow.xml
            trinidad-config.xml
            web.xml
        Page Flows
        TrainStopSamplePage.jspx
Application Resources
Data Controls
Recently Opened Files
```

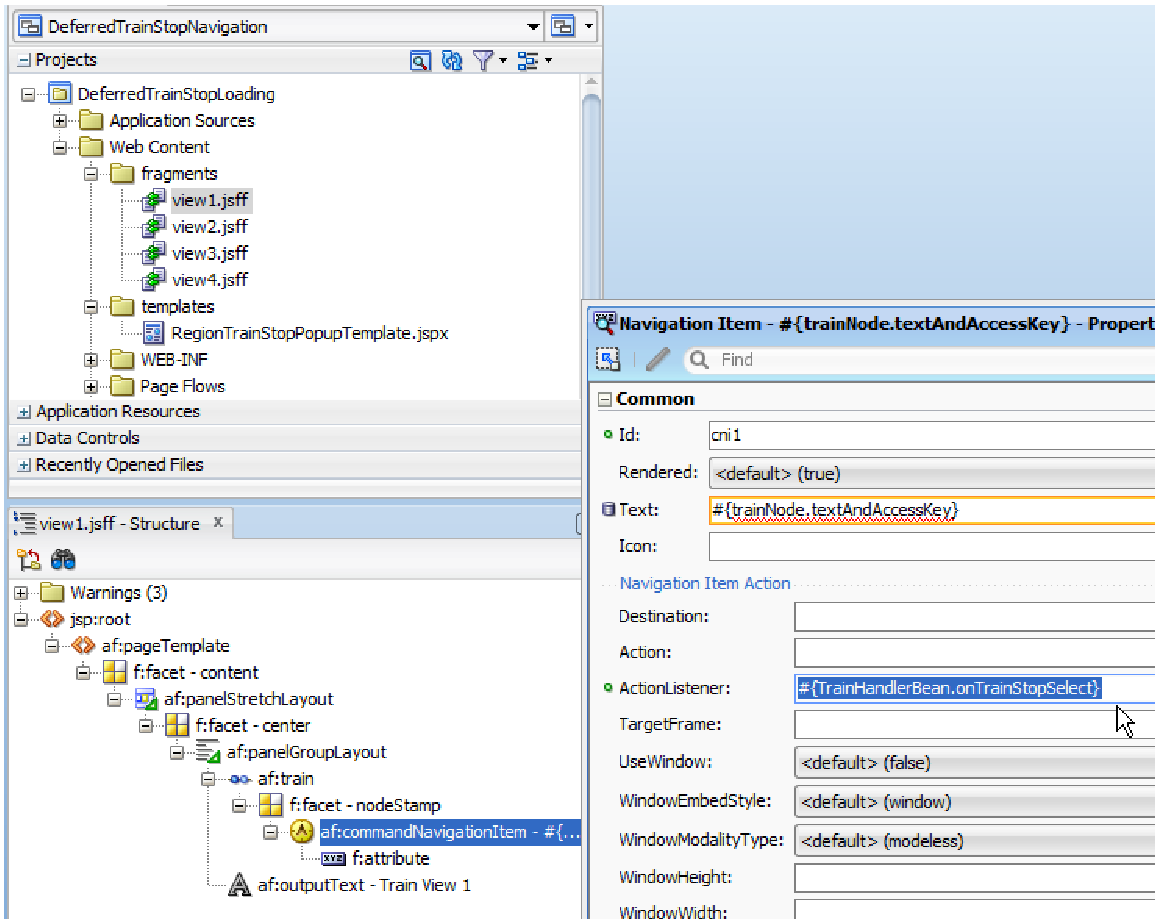
Using a page template is a convenient and time saving option to add a popup dialog to views without coding the popup into each view. In JDeveloper 11g R2, page templates can be nested, which allows you to reference functional templates – like this containing a popup to share – with page layout templates.



Each view contains an **af:train** component that in its **nodeStamp** facet has a **commanNavigationItem** component defined. Using a **commanNavigationItem** in the **nodeStamp** allows developers to change the default train rendering and behavior. The **commanNavigationItem ActionListener** property is set up to point to a managed bean method. Each view and train stop references the same managed bean method.

The managed bean method's responsibility is

- To determine the train stop the user clicked on
- Show the popup dialog (or if the use case is to conditionally show the popup, to determine the condition)


DeferredTrainStopNavigation

Projects

- DeferredTrainStopLoading
  - Application Sources
  - Web Content
    - fragments
      - view1.jsff
      - view2.jsff
      - view3.jsff
      - view4.jsff
    - templates
      - RegionTrainStopPopupTemplate.jspx
    - WEB-INF
    - Page Flows

Application Resources

Data Controls

Recently Opened Files

view1.jsff - Structure

- Warnings (3)
- jsp:root
  - af:pageTemplate
    - f:facet - content
      - af:panelStretchLayout
        - f:facet - center
          - af:panelGroupLayout
            - af:train
              - f:facet - nodeStamp
                - af:commandNavigationItem - #{...
                  - f:attribute
            - af:outputText - Train View 1

Navigation Item - #{trainNode.textAndAccessKey} - Propert

Common

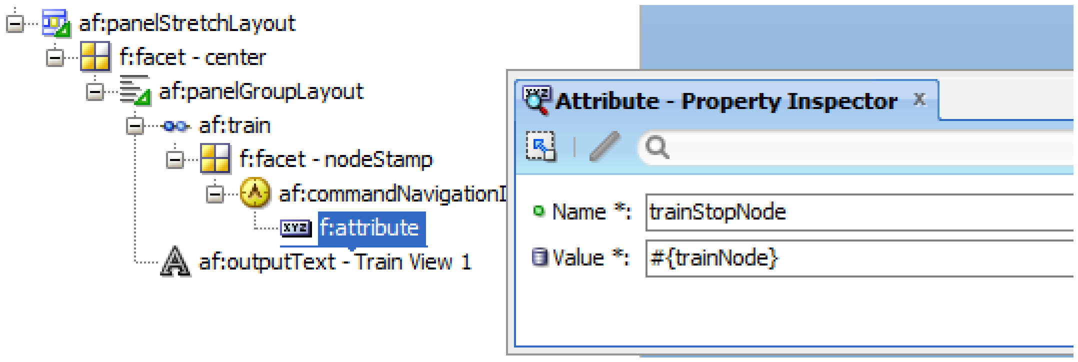
Id:	cni1
Rendered:	<default> (true)
Text:	#{trainNode.textAndAccessKey}
Icon:	

Navigation Item Action

Destination:	
Action:	
ActionListener:	#{TrainHandlerBean.onTrainStopSelect}
TargetFrame:	
UseWindow:	<default> (false)
WindowEmbedStyle:	<default> (window)
WindowModalityType:	<default> (modeless)
WindowHeight:	
WindowWidth:	

The **commanNavigationItem** has an **f:attribute** component added to keep track of which train stop the user clicked on. The **f:attribute** item adds a custom attribute to the ADF Faces component that can be accessed in the managed bean. The #{trainNode} reference is of type `TaskFlowTrainStopModel` and exposes the train stop navigation case by its `getOutcome()` method.

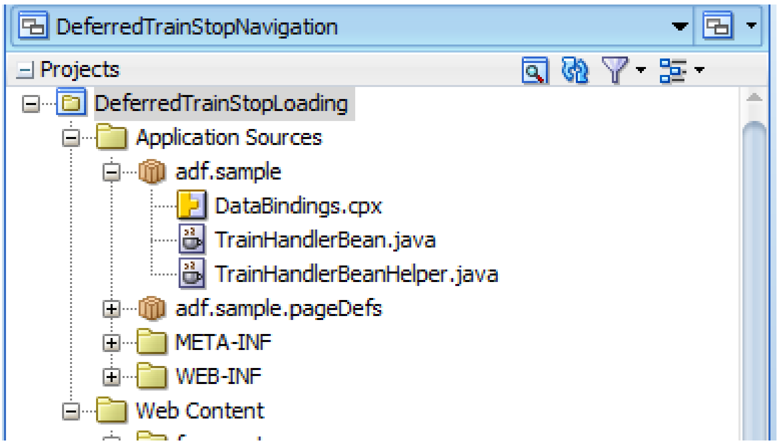
The **commanNavigationItem** instance is accessible from the managed bean method using the `ActionEvent` object that is passed to it.


- af:panelStretchLayout
  - f:facet - center
    - af:panelGroupLayout
      - af:train
        - f:facet - nodeStamp
          - af:commandNavigationI
            - f:attribute
      - af:outputText - Train View 1

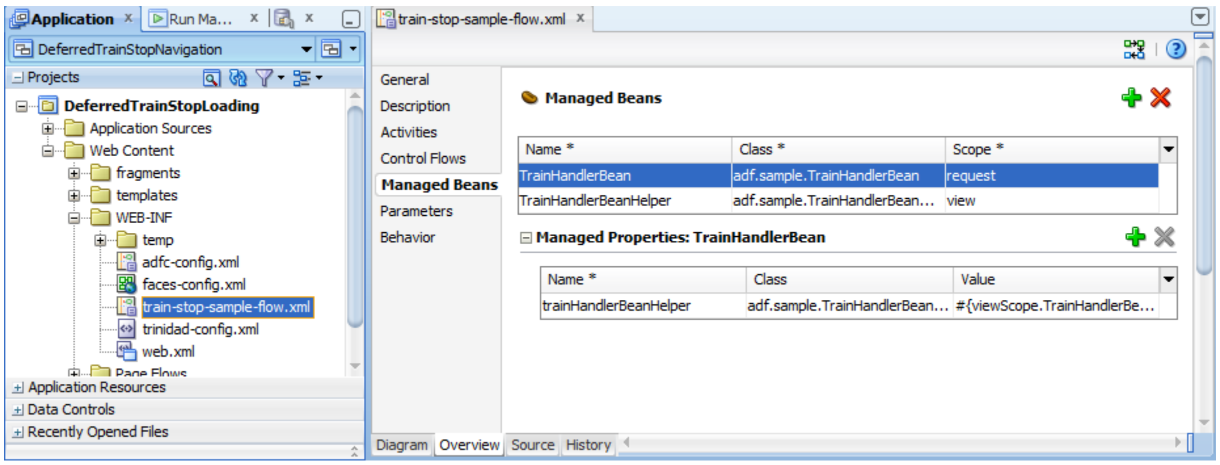
Attribute - Property Inspector

Name *:	trainStopNode
Value *:	#{trainNode}

Two managed beans are used in this sample. The **TrainHandlerBean** is defined in request scope and is referenced from the **commanNavigationItem** in the af:train component on the views and the af:popup component in the template. The **TrainHandlerBeanHelper** is configured in **viewScope** and holds the reference to the outcome value of the train stop clicked by the user. The **TrainHandlerBeanHelper** is configured as a managed property in the **TrainHandlerBean** definition. Both beans are configured in the bounded task flow metadata.



Shown below is the managed bean configuration for the **TrainHandlerBean**. As you can see, the **TrainHandlerBeanHelper** bean is referenced as a managed property, which is required for the popup listener that notifies the managed bean about the dialog close by the user to get access to the user selected train node.



Managed Beans

Name *	Class *	Scope *
TrainHandlerBean	adf.sample.TrainHandlerBean	request
TrainHandlerBeanHelper	adf.sample.TrainHandlerBean...	view

Managed Properties: TrainHandlerBean

Name *	Class	Value
trainHandlerBeanHelper	adf.sample.TrainHandlerBean...	#{viewScope.TrainHandlerBe...

The two managed bean code listings are shown below. I commented the code source so you know what each method therein does and from where it is accessed.

TrainHandlerBean (request scope)

The **TrainHandlerBean** implements all of the logic. It saves the user selected train stop node's outcome value (the implicit navigation case) in the **TrainHandlerBeanHelper** so the information persists beyond the request. The bean also opens the popup dialog by searching within the ADF region container for the af:popup instance.

When the popup is closed by the user pressing **Ok** or **Cancel**, the managed bean is called again. If the selected outcome is **Ok**, then the bean queues an action event on the ADF region to navigate to the next view.

```
package adf.sample;


import javax.el.ELContext;
import javax.el.ExpressionFactory;
import javax.el.MethodExpression;
import javax.faces.component.UIComponent;
import javax.faces.context.FacesContext;
import javax.faces.event.ActionEvent;
import javax.faces.event.PhaseId;
import oracle.adf.controller.TaskFlowTrainStopModel;
import oracle.adf.view.rich.component.rich.RichPopup;
import oracle.adf.view.rich.component.rich.fragment.RichPageTemplate;
import oracle.adf.view.rich.component.rich.fragment.RichRegion;
import
oracle.adf.view.rich.component.rich.nav.RichCommandNavigationItem;
import oracle.adf.view.rich.event.DialogEvent;

public class TrainHandlerBean {
  /**
   * Managed bean property reference to a helper bean in view scope
   */
    private TrainHandlerBeanHelper trainHandlerBeanHelper = null;

    public TrainHandlerBean() {
    }


    /**
     * Action method that is referenced from the train stop command
     * action to defer train navigation and allow developers to
     * interact with the user. In this sample a popup  dialog is opened
     * for the user to confirm train navigation.
     *
     * The use case for deferred trains top navigation includes manual
     * complex field validation, to check for
     * uncommitted data, e.g.
     * ControllerContext.getInstance().getCurrentViewPort()
     * .isDataDirty() , etc.
     *
     * @param actionEvent
     */

     public void onTrainStopSelect(ActionEvent actionEvent) {
       RichCommandNavigationItem rni =
            (RichCommandNavigationItem)actionEvent.getSource();
       TaskFlowTrainStopModel selectedTrainStop =
           (TaskFlowTrainStopModel)rni.getAttributes()
                                .get("trainStopNode");
       String outcome = selectedTrainStop.getOutcome();
       trainHandlerBeanHelper.setSelectedTrainStopOutcome(outcome);

       /*
        * ADD YOUR DEFERRED ACTION. FOR EXAMPLE, LAUNCH POPUP. TO
        * CONTINUE THE TRAIN NAVIGATION, CALL
        * queueTrainStopEventToRegion( ... )
       */

        //launch popup. Start search from af:region. The af:region and
        //the af:pageTemplate tags are
        //naming container. This has an impact to the runtime ID of the
        //popup component. So to find  popup component just by its ID
        //"pt_p1", we enter the region container, then the template to
        //then call findComponent("pt_p1")
        RichRegion adfRegion = this.findRichRegionContainer(rni);
        //in this sample, the popup component is in a template. Page
        //Fragments can only have a single
        //root component (everything else is flagged as an error) so
        //that the template must be the first children in the region

        RichPageTemplate regionTrainstopPopupTemplate = null;
        regionTrainstopPopupTemplate =
                (RichPageTemplate) adfRegion.getChildren().get(0);
        UIComponent component =
                 regionTrainstopPopupTemplate.findComponent("pt_p1");

        if(component != null){
          RichPopup richPopup = (RichPopup) component;
          //align popup to screen center
          RichPopup.PopupHints hints = new  RichPopup.PopupHints();
          richPopup.show(hints);
        }
        else{
            //TODO change to use logger
            System.out.println("The popup instance with ID pt_p1 could
                                 not be found");
        }
    }

    /**
     * Method that finds the af:region container for a bounded task
     * flow exposed in a region. Don't use this method outside of ADF
     * regions.
     * @param uiComponent Component within the region that is the
```

```
     * starting point for the search
     * @return The RichRegion instance as UIComponent
     */
    private RichRegion findRichRegionContainer(UIComponent uiComponent)
    {
      UIComponent currentComponent = uiComponent;
        while(!(currentComponent instanceof RichRegion)){
           //task flows in a region always have a RichRegion container
           //sonewhere.
           currentComponent = currentComponent.getParent();
         }
        return (RichRegion) currentComponent;
    }


   /**
    * In this sample the popup contains a dialog with OK, Cancel
    * button.
    * When OK is clicked, the train stop navigation should progress. If
    * not, the train stop should not continue and remain on the current
    * page
    *
    * @param dialogEvent
    */
    public void onDialogAction(DialogEvent dialogEvent) {
      //only if user confirmed navigation to the next train stop,
      //perform navigation. Otherwise ignore request and remain
      //on current train stop view
      if (dialogEvent.getOutcome() == DialogEvent.Outcome.ok) {
        //perform saved train stop navigation
        queueTrainStopEventToRegion("#{viewScope.TrainHandlerBeanHelper
                                     .getSelectedTrainStopOutcome}",
                                     dialogEvent.getComponent());
      }
    }

    /**
     * Wrapper method around the RichRegion queueActionEventInRegion
     * API.
     * @param outcomeEL The EL to access a managed bean that returns
     * the navigation string to to follow. This can be an outcome for a
     * navigation case, or as in this sample, the implicit navigation
     * case used in trains
     *
     * @param searchComponentInRegion To queue the action, we need a
     * handle to the RichRegion component. Providing us with a
     * component residing in a region is all that is
     * needed to search the af:region container
     */
    private void queueTrainStopEventToRegion(String outcomeEL,
                         UIComponent searchComponentInRegion) {
      //This sample assumes bounded task flows to be exposed in an
      //af:region. So it is safe to assume a parent component to be of
      //type RichRegion. Let's find it to queue the train stop
      //outcome event for navigation.
      RichRegion adfRegion = null;

      adfRegion =
          this.findRichRegionContainer(searchComponentInRegion);
      FacesContext fctx = FacesContext.getCurrentInstance();
      ExpressionFactory expressionFactory =
                  fctx.getApplication().getExpressionFactory();
     ELContext elctx = fctx.getELContext();
     MethodExpression methodExpression =
         expressionFactory.createMethodExpression(elctx,
                                                  outcomeEL,
                                                  String.class,
                                                  new Class[] { });
       //queue action in region
       adfRegion.queueActionEventInRegion(
                   methodExpression, null, null, false, 0, 0,
                   PhaseId.INVOKE_APPLICATION);
    }

    /**
     * Managed bean property reference to a managed bean in view scope
     * @param trainHandlerBeanHelper Managed bean inserted as a managed
     * property reference
     */
    public void setTrainHandlerBeanHelper(
      TrainHandlerBeanHelper trainHandlerBeanHelper) {
       this.trainHandlerBeanHelper = trainHandlerBeanHelper;
    }

    public TrainHandlerBeanHelper getTrainHandlerBeanHelper() {
        return trainHandlerBeanHelper;
    }
}
```

TrainHandlerBeanHelper (view scope)

This managed bean is just a helper class that saves the selected train node outcome value for later navigation when the user confirms the popup dialog with **Ok**.

```
package adf.sample;

public class TrainHandlerBeanHelper {
  private String selectedTrainStopOutcome = null;
  public TrainHandlerBeanHelper() {
      super();
  }

  public void setSelectedTrainStopOutcome(
                       String selectedTrainStopOutcome) {
    this.selectedTrainStopOutcome = selectedTrainStopOutcome;
  }

   public String getSelectedTrainStopOutcome() {
     return selectedTrainStopOutcome;
   }
}
```

Sample Download

The Oracle JDeveloper 11.1.2.1 sample workspace is available as sample 97 from ADF Code Corner.

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

Though the sample is developed with JDeveloper 11g R2, the solution also works with previous versions of Oracle JDeveloper 11g.

RELATED DOCOMENTATION

⊠	Oracle Magazine article about trains http://www.oracle.com/technetwork/issue-archive/2011/11-sep/o51adf-452576.html
⊠	Trains in the ADF product documentation http://docs.oracle.com/cd/E16162_01/web.1112/e16182/taskflows_complex.htm#CJHFBFIE
⊠	Related ADF Code Corner articles http://www.oracle.com/technetwork/developer-tools/adf/learnmore/93-differentuifortrainstops-1413952.pdf http://www.oracle.com/technetwork/developer-tools/adf/learnmore/82-programmatically-navigate-trains-396873.pdf http://www.oracle.com/technetwork/developer-tools/adf/learnmore/80-dyn-sequential-config-train-387002.pdf
