ORACLE

ADF Architecture Square

ADF Region Interaction: External Train Navigation

twitter.com/adfArchSquare

Abstract

The ADF bounded task flow train model is an alternative to control flow cases for users to navigate views in bounded task flows. Bounded task flows are specifically designed to have one entry point, and as a result represent isolated and encapsulated units of work. For example the train model contained within a bounded task flow is not directly accessible from the outside, it can only be accessed via the single entry point of the bounded task flow. A frequent requirement for bounded task flows exposed in ADF regions though is to allow the parent view or a contained child region to navigate the bounded task flow using the train model.

This article explains a pattern that allows you to navigate bounded task flow trains from outside of the task flow.

Author: Frank Nimphius
Date: 30/SEP/2012

Introduction

A bounded task flow can be configured to expose a train model to navigate from one view to another, just by setting the task flow's **Train** property. In the view activities of the task flow, you use the Oracle ADF Faces **af:train** and **af:trainButtonBar** components to render the train model in the UI. The view activities itself must have its **Train Stop** property set to true to become a stop in the train model.

At runtime users navigate to a view represented by a train stop by clicking on the representative train stop icon. Alternatively, developers can choose to programmatically navigate a train model in response to a user action for example.

This article focuses on a special use case of programmatic navigation, that is to make a bounded task flow train model contained within a region navigate by an external programmatic call.

The technique explained in this article is non intrusive in that it does not break the encapsultaion of the bounded task flow. Instead you expose a custom API for the parent view or a contained child task flow to use for navigating the bounded task flow. Because the pattern is generic and does not maintain any dependencies to the bounded task flow it is implemented in, it's a good candidate to put into a bounded task flow template for reuse in multiple task flows.

Note: Working with the train model doesn't require a train component to be added to a view . If programmatic train navigation is sufficient then no UI (af:train, af:trainButtonbar) needs to be provided for the train model in a view.

Pattern Configuration and Implementation

The core implementation code of the **External Train Navigation** pattern for region interaction mainly exists within the bounded task flow to navigate. The implementation on the parent view and child region, also explained in the following document, purely shows how to work with the exposed train navigation API.

External Task Flow Navigation Pattern: Bounded Task Flow Implementation

Beside the train model configuration – which is out of scope for this article – the bounded task flow you want to navigate from an external needs to be configured with two method activities associated with a wildcard navigation control flow as shown in figure 1.

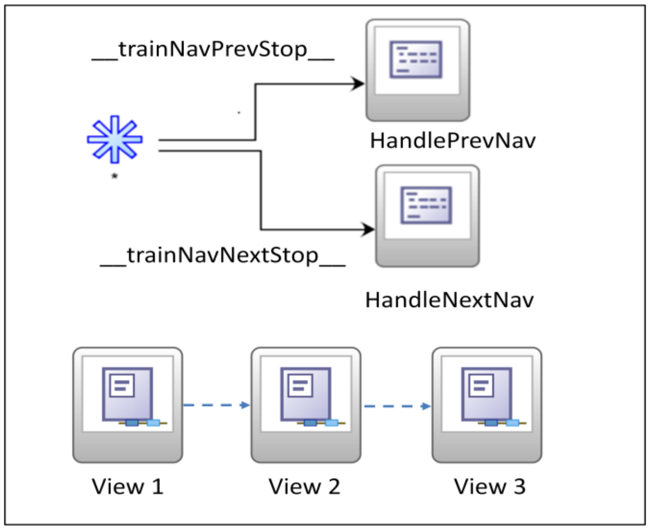


Figure 1: Bounded task flow with method activities

Figure 1 shows the two method activities above the three views to navigate. The control flow names in this example are somewhat cryptic within underscores ("__trainNavPrevStop__" and "__trainNavNextStop__") but help to avoid control flow naming conflicts in case you implement this solution in a bounded task flow template for reuse.

To externally trigger navigation in the bounded task flow, you queue the control flow case of one of the method activities, "__trainNavPrevStop__" or "__trainNavNextStop__", on the parent view or the child region.

Shown in figure 2, the method activities associated with the control flows reference a managed bean method. The managed bean must be configured in the same bounded task flow (or the bounded task flow template) that contains the method activities and should have backing bean scope.

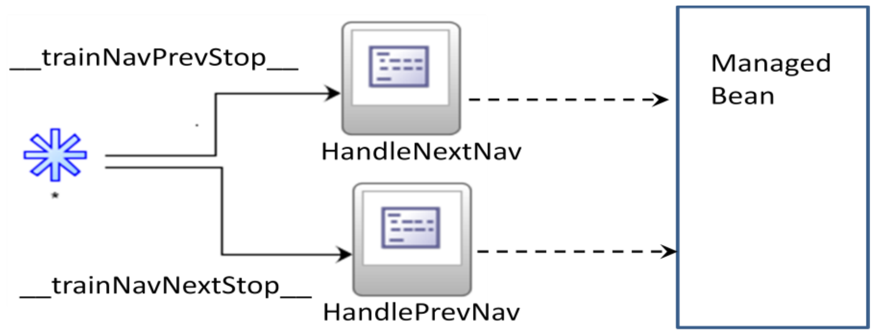
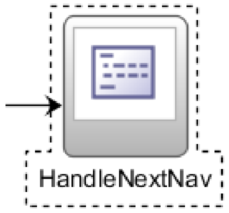


Figure 2: Method activities are configured to access managed bean methods in task flow

Figure 3, shows the configuration of one of the method activities. In this sample, the **Method** property references the "`onTaskFlowNavPrev`" method of the managed bean, to look up the current train model state to determine the next enabled train stop. As train navigation is based on implicit created control flow cases, if such a control flow case name is returned from a method activity, navigation will happen to the target view.



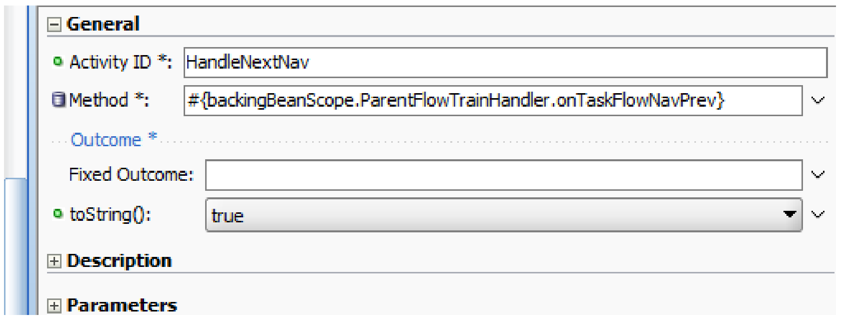

General
Activity ID *: HandleNextNav
Method *: #{backingBeanScope.ParentFlowTrainHandler.onTaskFlowNavPrev}
Outcome *
Fixed Outcome:
toString(): true
Description
Parameters

Figure 3: Method activity configuration

Node the setting of the **toString()**propery of the method activity in figure 3.

For a method activity to perform navigation you either need to define a value for the **Fixed Outcome** property or set the **toString()**to true. If you don't set one of these, or both, you create invalid XML and the task flow cannot be run. Setting **troString()** to true will convert the return value of the method call into a string and use this for navigation. This is the setting needed for the pattern explained in this document.

The sample code below demonstrates for the managed bean of the bounded task flow how to programmatically navigate the train. One method performs the train forward navigation, while the other handles the back navigation. Before navigating to a next or previous stop, a check is performed to determine if the target node is enabled or disabled (controlled by the view activity **skip** property). If so then the next stop is tried for navigation.

Disclaimer: The code uses `TrainStopModel`, which is an internal framework class that should not be used in custom applications. A bug is filed to make this class public so it can be used with no precaution. Until this public class is implemented it is easiest to go with the internal class instead of working around the missing functionality.

```
 /**
   * Method navigates to the next enabled train stop if any.
    * @return implicit outcome of the train stop
*/

public String onTaskFlowNavNext() {
 String nextStopAction = null;
 ControllerContext controlCtxt = ControllerContext.getInstance();
 ViewPortContext viewPortCtxt  = controlCtxt.getCurrentViewPort();
 TaskFlowContext taskFlowCtxt  = viewPortCtxt.getTaskFlowContext();
 TaskFlowTrainModel trainModel = taskFlowCtxt.getTaskFlowTrainModel();
 TaskFlowTrainStopModel currentStop = trainModel.getCurrentStop();
 TrainStopModel nextStop =
           (TrainStopModel)trainModel.getNextStop(currentStop);
 while (nextStop != null) {
  if (nextStop.isEnabled()) {
   nextStopAction = nextStop.getOutcome();
   break;
  }
 nextStop = (TrainStopModel)trainModel.getNextStop(nextStop);
 }
 return nextStopAction;
}

/**
 * Method navigates to the previous enabled train stop if any.
 * @return implicit outcome of the train stop
 */
public String onTaskFlowNavPrev() {
 String prevStopAction = null;
 ControllerContext controlCtxt = ControllerContext.getInstance();
 ViewPortContext viewPortCtxt  = controlCtxt.getCurrentViewPort();
 TaskFlowContext taskFlowCtxt  = viewPortCtxt.getTaskFlowContext();
 TaskFlowTrainModel trainModel = taskFlowCtxt.getTaskFlowTrainModel();
 TaskFlowTrainStopModel currentStop = trainModel.getCurrentStop();
 TrainStopModel prevStop =
       (TrainStopModel)trainModel.getPreviousStop(currentStop);

 while (prevStop != null) {
  if (prevStop.isEnabled()) {
   prevStopAction = prevStop.getOutcome();
   break;
  }
  prevStop = (TrainStopModel)trainModel.getPreviousStop(prevStop);
 }
 return prevStopAction;
}
```

To summarize the work you need to complete in the bounded task flow is:

1. Optional, but recommended, create a bounded task flow template to create a reusable implementation of this technique
2. Enable the train model on the bounded task flow
3. Configure views as train stops, which is automatic for all views created after enabling the train model for a bounded task flow
4. Create two wild card navigation cases to method activities
5. Create a managed bean with the code shown above or a custom version of it
6. Document the names of the wildcard flows for developers who want to use this API

Configuration: Parent View That Triggers Navigation

Indicated in figure 4, the **af:region** tag in the parent view that exposes the bounded task flow is represented by an instance of `RichRegion` at runtime.

The `RichRegion` class has a public method – queueActionEventInRegion – that you use to invoke a control flow case on the bounded task flow. To navigate the train model you need to queue an action for either the "__trainNavPrevStop__" or "__trainNavNextStop__" control flow cases. Remember that the two control flow cases navigate to a method activity in the bounded task flow to determine the next navigation target. If a valid target is found, navigation occurs.

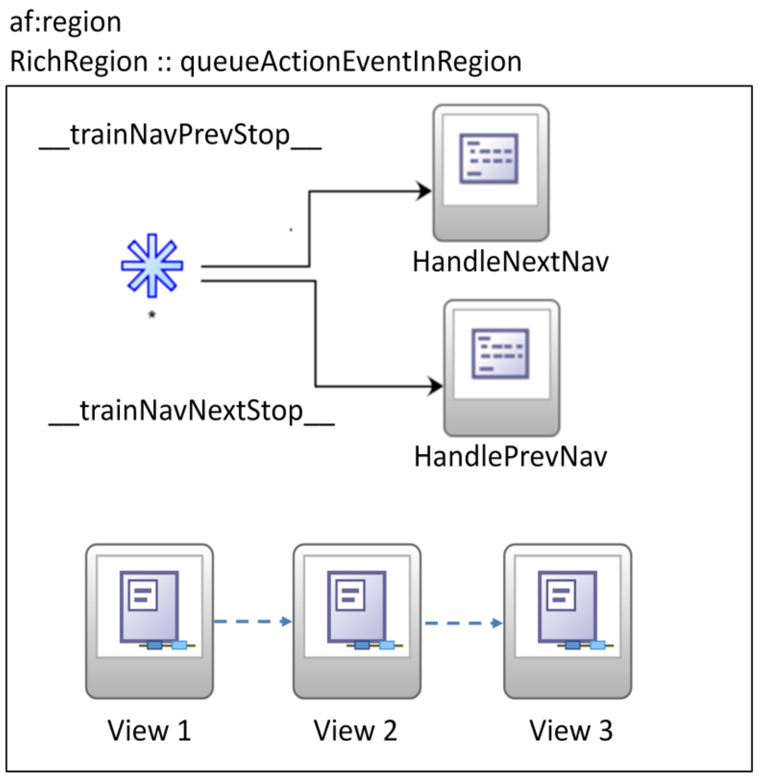


Figure 4: location of RichRegion queueActionEventInRegion method

To achieve this use the following code in a managed bean and reference it from command buttons for forward or backward navigation of the bounded task flow train model. Note that in the sample code, the reference to the **af:region** instance is created through the af:region **Binding** property.

```
// Method to reference from a command item for next navigation
public String onNavNext() {
 queueActionEvent("__trainNavNextStop__");
 return null;
}
```

```
// Method to reference from a command item for backward navigation
public String onNavPrev() {
 queueActionEvent("__trainNavPrevStop__");
 return null;
}
```

```
// Nethod to queue navigation on the region
private void queueActionEvent(String action) {
 FacesContext fctx = FacesContext.getCurrentInstance();
 ExpressionFactory ef = fctx.getApplication().getExpressionFactory();
 ELContext ectx = fctx.getELContext();
 MethodExpression me = ef.createMethodExpression(ectx, action,
                           String.class, new Class[] { });
 regionContainer.queueActionEventInRegion(me, null, null,
                                               false, -1, -1,
 PhaseId.INVOKE_APPLICATION);
}


// Setter/getter methods created by JDeveloper when creating the
// af:region Binding reference
public void setRegionContainer(RichRegion regionContainer) {
 this.regionContainer = regionContainer;
}
public RichRegion getRegionContainer() {
 return regionContainer;
}
```

Note: if you perform train navigation from command button or link, ensure partialSubmit property is set to true to avoid a full page refresh

To summarize the work you need to complete in the parent view to trigger train navigation in the bounded task flow is:

1. Create command items for forward or backward train navigation
2. Create a managed bean in the parent view task flow and create two action methods and one binding reference to the af:region
3. Call queueActionEventInRegion on the af:region reference, passing the control flow case names for the method activities in the bounded task flow to navigate the train

Configuration: Child Region That Triggers Navigation

To navigate a bounded task flow from a child region contained with the bounded task flow, you use the ADF task flow parent action activity you find in the JDeveloper component palette for the task flow diagrammer.

The parent action takes a control flow name – "__trainNavPrevStop__" or "__trainNavNextStop__" in this example – and invokes it on either the parent region, or the root region. Figure 5 shows the use case: View 2 contains a child region, from where you want to navigate to either View 1 or View 2 in the parent task flow.

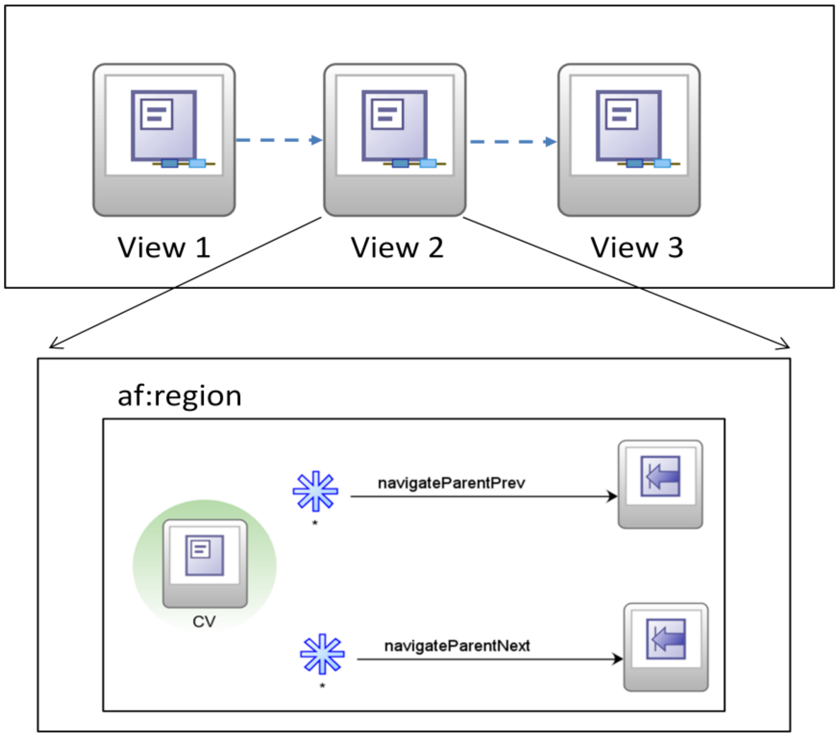
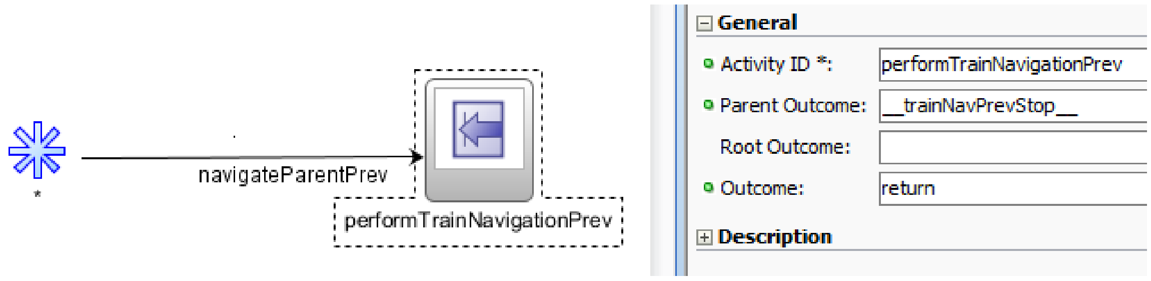


Figure 5: Bounded task flow with a view containing a child region

The bounded task flow in the child region has two parent actions defined, one for backward and one for forward navigation of the parent task flow train model. The parent action is navigated to from command items, like buttons, menus or a link. The use of a wild card navigation as shown in figure 5 is optional and allows any view in a child task flow to navigate the parent flow.



Activity ID *:	performTrainNavigationPrev
Parent Outcome:	__trainNavPrevStop__
Root Outcome:	
Outcome:	return

Figure 6: Parent action configuration

The parent action has its **Parent Outcome** property configured with the name of a wild card control flow in the parent bounded task flow. The **Root Outcome** property can be used to navigate the top level task flow, which is the browser view. The **Outcome** property defines a possible callback if parent navigation fails. In the example, nothing should happen in this case.

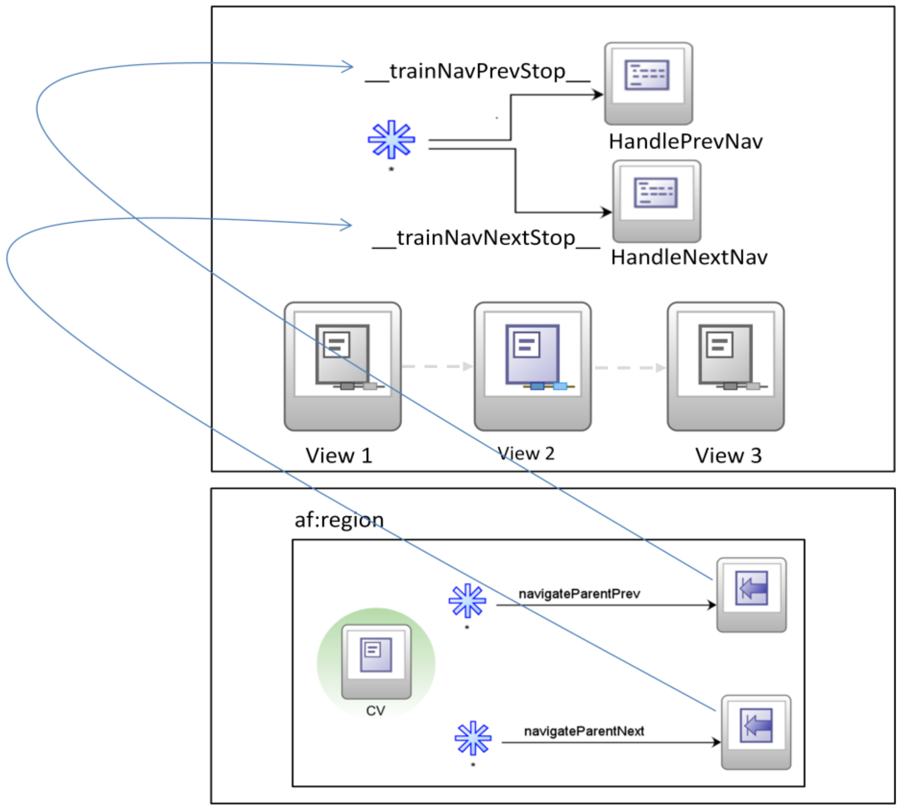


Figure 7: Parent action in action

Figure 7 shows how the child region call to the parent task flow works to trigger train navigation. Each parent activity in the child region references a control flow case leading to a method activity in the parent task flow to perform train navigation. Because the method activities execute within the parent task flow, the train model navigation command is executed in the right context.

To summarize the work you need to complete

1. Add parent activity elements to be child task flow that should trigger train model navigation on the parent task flow
2. Configure the **Parent Outcome** property to point to a control flow case that exists in the parent task flow. The parent task flow control flow case should then navigate to a method call activity that navigates the train model
3. Define a value for the **Outcome** property so that navigation finds a fallback navigation within the child region in case parent navigation fails (which would indicate a design time issue in setting up the **Parent Outcome** property)

Summary

This article outlines two related techniques to navigate bounded task flows exposed in a region from either the parent view or a child region. Both techniques use the same managed bean code within the navigated task flow. The API that is created for consumers of such an implementation is the name of a control flow case.

RELATED DOCOMENTATION

☒	Oracle product documentation about trains http://docs.oracle.com/cd/E23943_01/web.1111/b31974/taskflows_complex.htm#CJHFBFIE
☒	af:train tag documentation http://docs.oracle.com/cd/E23943_01/apirefs.1111/e12419/tagdoc/af_train.html
☒	All aboard – Oracle Magazine article about train navigation http://www.oracle.com/technetwork/issue-archive/2011/11-sep/o51adf-452576.html
☒	How to programmatically navigate train models http://www.oracle.com/technetwork/developer-tools/adf/learnmore/82-programmatically-navigate-trains-396873.pdf
☒	Put a different look to your train stops http://www.oracle.com/technetwork/developer-tools/adf/learnmore/93-differentuifortrainstops-1413952.pdf
☒	Dynamically enabling or disabling train stops http://www.oracle.com/technetwork/developer-tools/adf/learnmore/80-dyn-sequential-config-train-387002.pdf