ADF Code Corner

82. How to programmatically navigate ADF trains

twitter.com/adfcodecorner

Abstract:

Optionally, ADF bounded task flows can be configured to expose a train model to their contained views for users to navigate between views and task flow calls using a train component. Usually navigation with train components is triggered by the user clicking on a train stop in a train bar or clicking a button in a trainButtonBar. Navigation in bounded task flow should be consistent and follow the same rukes, no matter if it is triggered by a user action or programmatically invoked. This article explains how to use the TaskFlowTrainStop API to prgrammatically navigate trains.

Author: Frank Nimphius, Oracle Corporation
twitter.com/fnimphiu
10-MAY-2011

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.

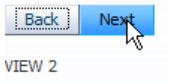
Please post questions or report problems related to the samples in this series on the OTN forum for Oracle JDeveloper: http://forums.oracle.com/forums/forum.jspa?forumID=83

Introduction

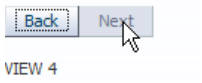
The pre-requisite for this article is for you to understand trains in ADF bounded task flows, as explained in the Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework.

A use case that could be handled by programmatic navigation of train models is to customize the behavior of the default train button bar. The train button bar shows to command buttons for back and forward navigation. It performs sequential navigation and stops when a train stop is disabled, which is the case when a train stop is configured to be skipped. In this case the train button bar only continues navigation when the skipped state on the train stop is changed to false.

If a use case requires the skipped train stop to be ignored and the navigation to proceed with the next enabled stop, then you need to create your own train button bar, using Java to navigate the train model.



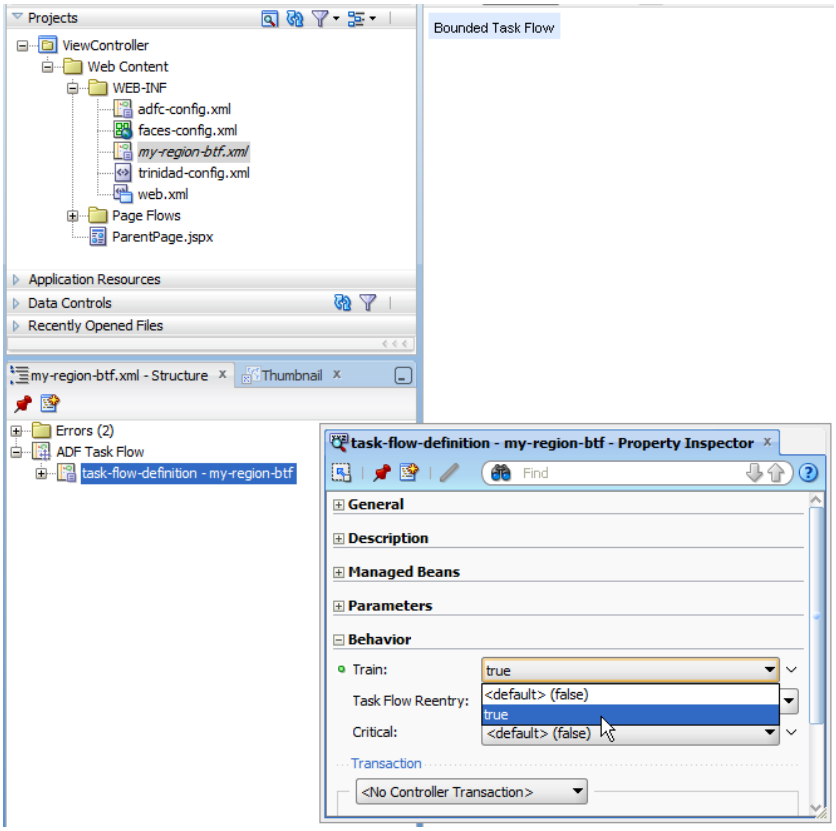
As shown in the image abobe and below, using a custom train button bar, you can determine the next navigation step after a skipped stop and continue with it. To build a custom train button bar, you only need to align to `af:commandButtons` using a panel PanelGroupLayout component.



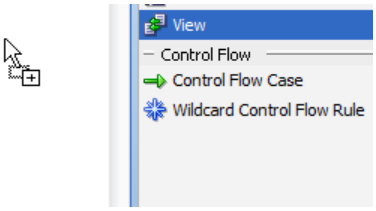
Another use case is to have a button that directly navigates to the last step in a train process, skippig all the stops between. Or, what about navigating the train model in response to a value change event or a table selection event?

ADF Trains: A Crash Course

Bounded task flows can be configured to expose a train model to the contained views. To expose the train model, you need to set the **Train** property on the bounded task flow definition to **true**.



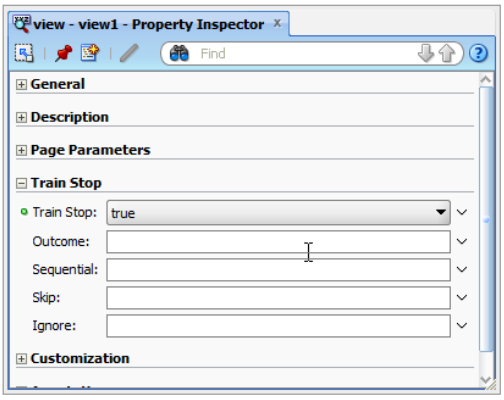
When you add views to a task flow that has its train model enabled, then all views are automatically added as a train stop, which visually is indicated by a dotted line between views and call activities. Existing views are added to the train by using the right mouse context menu for a selected activity.



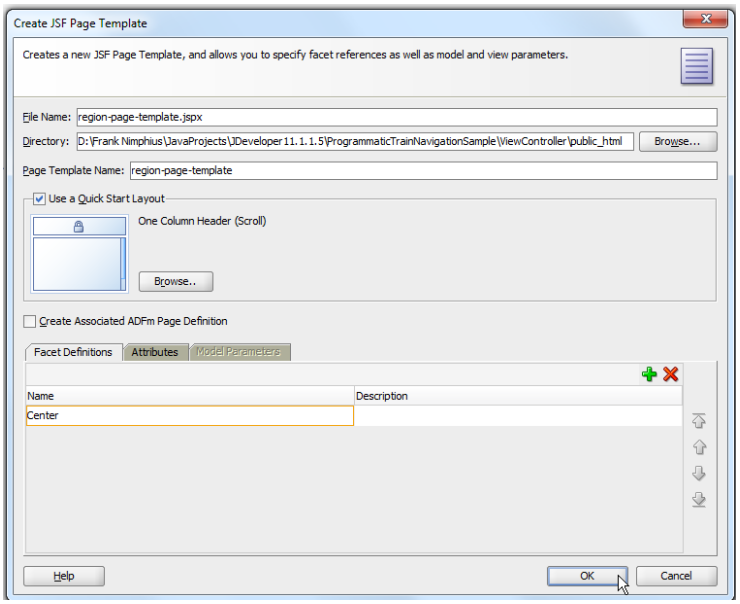
Each train stop in a train sequence can be configured using the Property Inspector. In this example you configure the **Skip** property to read its current value from a managed bean that extends HashMap. How to set this HashMap up is explained in sample 80 on ADF Code Corner:

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

The reason you need to have a HashMap controlling the **Skip** property state is because you need to be able to determine whether a train stop is skipped, and, if needed, be able to dynamically change its state.



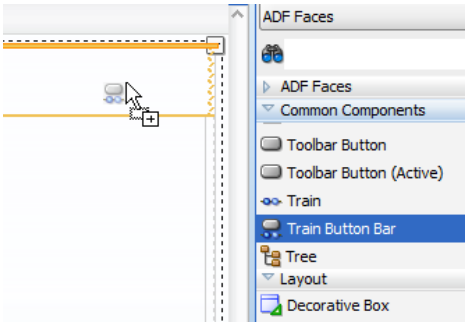
Trains are well suited to be added to the header of a page template that you can create as shown in the image below.



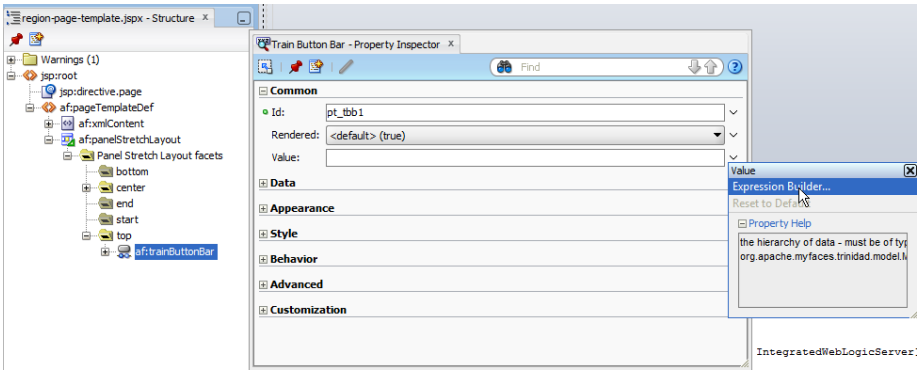
You can read more about page templates in the Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework

http://download.oracle.com/docs/cd/E17904_01/web.1111/b31973/af_reuse.htm#CACCFCJC

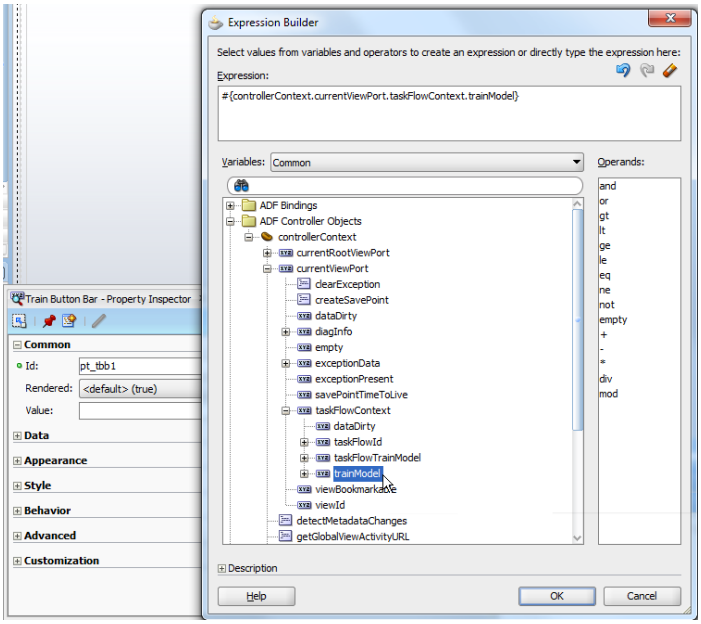
When adding a **Train** or **Train Button Bar** component to a page template, then you need to manually configure it to reference the `ControllerContext` for the task flow train model.



Select the train component, `af:trainButtonBar` in the example below, and open the Property Inspector. Browse to the **Value** property and choose **Expression Builder** from the context menu.

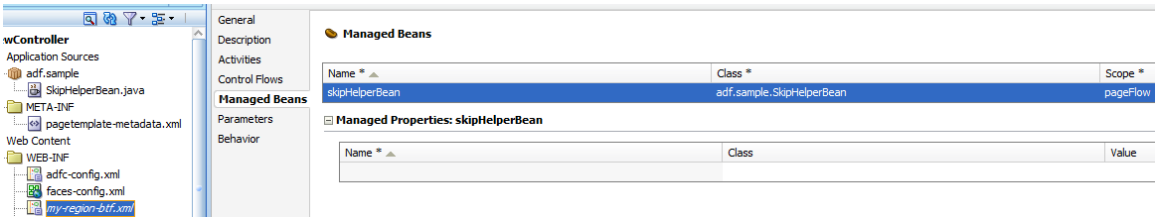


Expand the **ADF Controller Objects** node and select **currentViewPort | taskFlowContext | trainModel**



Using a HashMap for dynamic skip setting and change

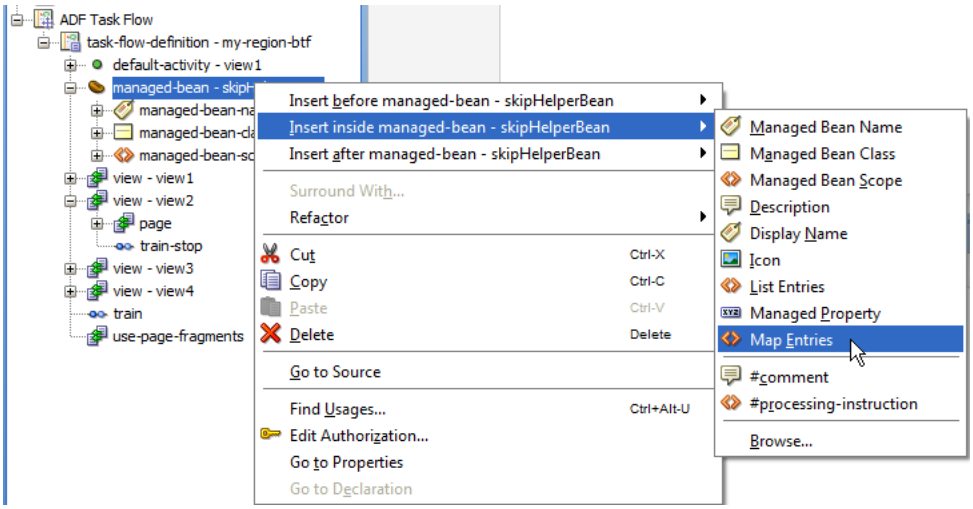
In the example you can download for this article, the value of the train-stop **Skip** property is determined for each stop by a managed bean. This managed bean needs to be configured in the task flow configuration of the bounded task flow.



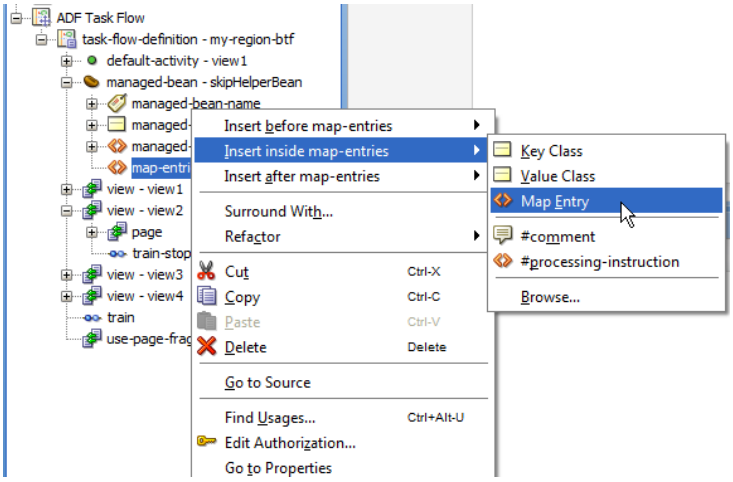
The Java code for the `SkipHelperBean` is shown below

```
import java.util.HashMap;

public class SkipHelperBean extends HashMap{
  public SkipHelperBean() {
     super();
  }

  @Override
  public Boolean get(Object key) {
    boolean skip = false;
    Object keyValue =  super.get(key);
    if(keyValue != null){
      //key found. Can we parse it to boolean?
      //TODO do you want to log the missing configuration ?
      try {
         skip = Boolean.parseBoolean((String)keyValue);
       } catch (Exception e) {
          //if it cannot be parsed, set skip to false
          skip = false;
          //TODO log failed parsing
       }
     }
      return skip;
  }
}
```

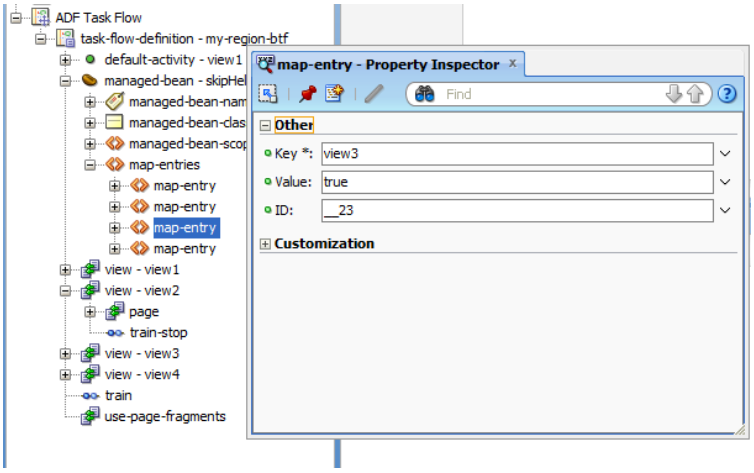
Because the state saved in the bean is used throughout the task flow, the bean is configured as a managed bean in the task flow configuration file with its scope set to **PageFlow**. After you define the bean, you select the bounded task flow file in the **Application Navigator** and open the **Structure Window**. In here, browse to the managed bean entry "skipHelperBean" to define the default setting for train stop's skip behavior. If you don't provide a default configuration, then no stop is skipped.



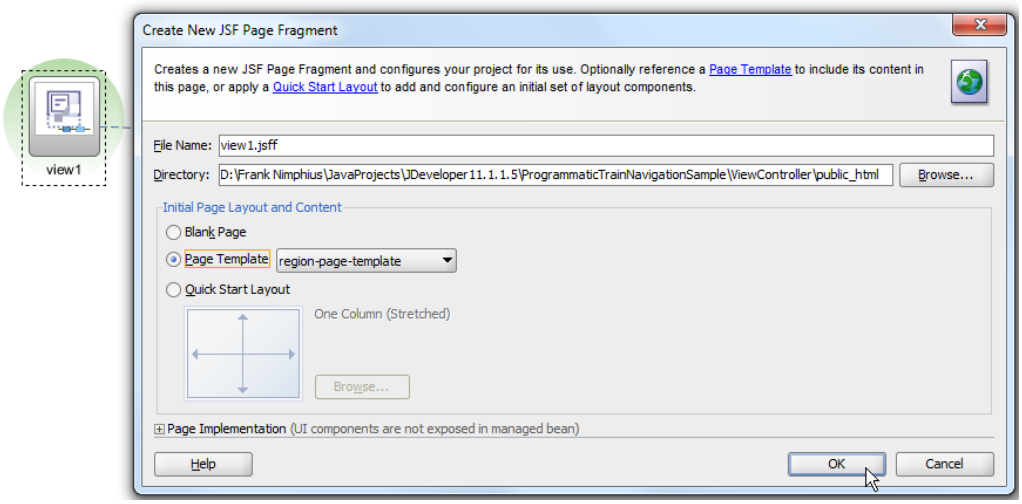
The default configuration is defined as **map-entries**. Map entries can be used for managed beans that are of type `HashMap` or that extend `HashMap`.



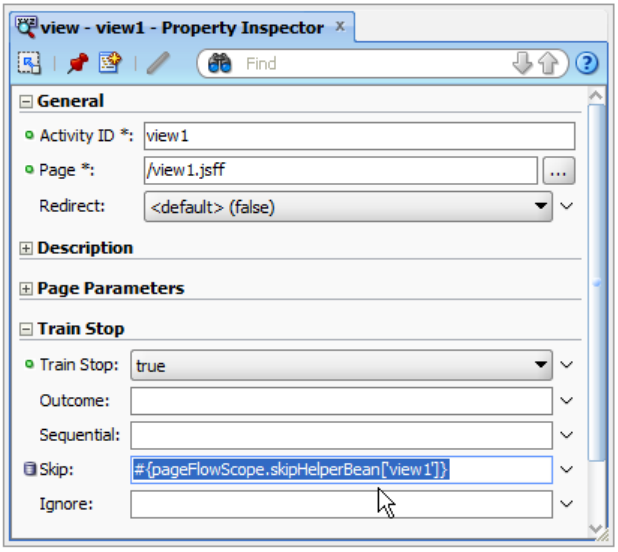
Add instance of **Map Entry** to the **map-entries** and use the Property Inspector to set the **Key** property to the activity Id of the view or task flow call. Set the **value** property to true (for skip) or false (for (no skip). The `HashMap` entries can be accessed for read and write at any time using Java or EL.



When creating new views in the bounded task flow, ensure they use the page template, **region-page-template** in the sample that contains the train UI components.



Next, for each train stop, configure the **Skip** property to point to the managed bean in **pageFlowScope**. The argument of the bean reference **['viewName']** is the activityId of the view or task flow call activity you configure. The EL expression reads the **true** or **false** value for the train stop when rendering the train UI.

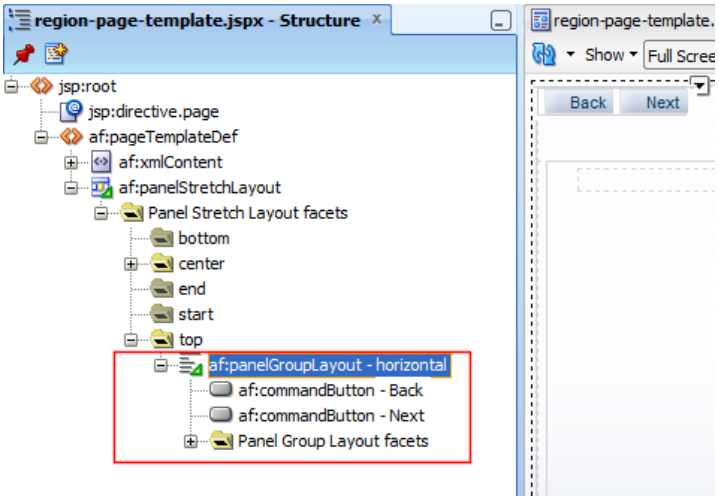


Until here, you did not change the default behavior of the `af:trainButtonBar` component, which is that it stops navigation when it reaches a train stop that has its **Skip** property set to false.

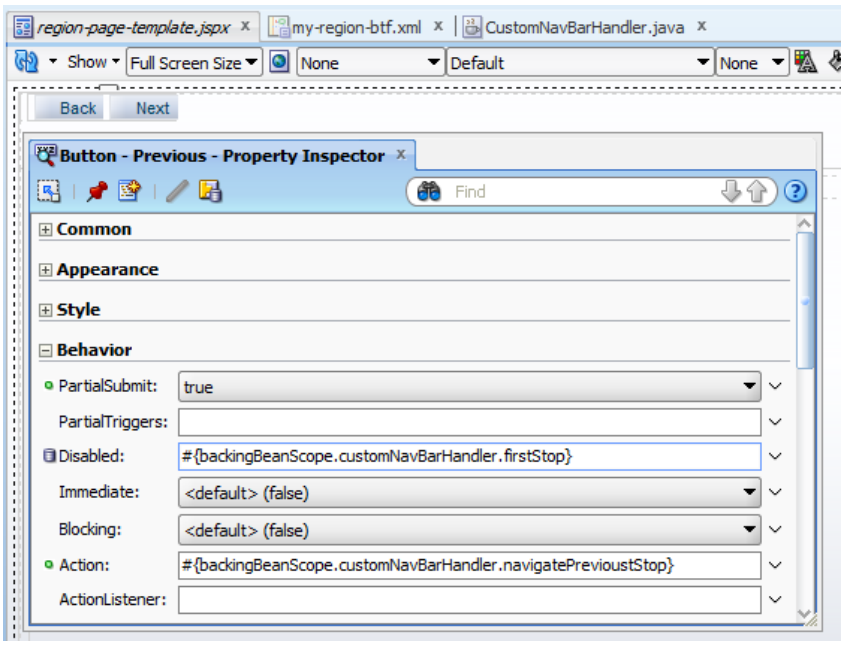
To fix this, you need to create a custom train button bar out of regular command items.

How-to navigate trains from component action events

The good thing with templates is that a change in a single location applies these changes to all pages that use the template. In this sample, the template is changed to use a custom pair of command buttons to replace the `af:trainButtonBar` component.



The **action** properties of the command buttons are configured to point to a managed bean in backing bean scope, as shown below



Note that the **Disabled** property too points to a method in the managed bean. This is to disable the "Back" button when the current stop is the first stop in a train, and the "Next" button when the current stop is the last.

Important! The custom train button bar is in a template and the template references a managed bean in backing bean scope. You need to make sure that the managed bean is configured in all task flows that have views using this template. A good naming convention will help when working in teams to avoid confusion.

The managed bean methods provide the following functionality

- Determine if the train stop is the first in the sequence. If so then true is returned
- Determine if the train stop is the last in the sequence. If so then true is returned
- Determine the outcome String of the next train stop after ensuring, using the HashMap bean, that the next stop is not skipped. If the next stop is skipped, try the next one until the last train stop in a series is reached.
- Determine the outcome String of the previous train stop after ensuring, using the HashMap bean, that the previous stop is not skipped. If the previous stop is skipped, try the one before until the first train stop in a series is reached.

```
import javax.el.ELContext;
import javax.el.ExpressionFactory;
import javax.el.ValueExpression;
import javax.faces.context.FacesContext;

import oracle.adf.controller.ControllerContext;
import oracle.adf.controller.TaskFlowContext;
import oracle.adf.controller.TaskFlowTrainModel;
```

```
import oracle.adf.controller.TaskFlowTrainStopModel;
import oracle.adf.controller.ViewPortContext;

public class CustomNavBarHandler {
    boolean firstStop = false;
    boolean lastStop = false;
    public CustomNavBarHandler() {
        super();
  }

  /**
    * Navigates to the next stop in a train
    * @return outcome string
    */
public String navigateNextStop() {
  String nextStopAction = null;
  ControllerContext controllerContext =
                           ControllerContext.getInstance();
  ViewPortContext currentViewPortCtx =
                           controllerContext.getCurrentViewPort();
  TaskFlowContext taskFlowCtx =
                           currentViewPortCtx.getTaskFlowContext();
  TaskFlowTrainModel taskFlowTrainModel =
                           taskFlowCtx.getTaskFlowTrainModel();
  TaskFlowTrainStopModel currentStop =
                           taskFlowTrainModel.getCurrentStop();

  TaskFlowTrainStopModel nextStop =
                           taskFlowTrainModel.getNextStop(currentStop);
  while(nextStop != null){
    if(isSkipTrainStop(nextStop) == false){
      //no need to loop any further
      nextStopAction = nextStop.getOutcome();
      break;
    }
    nextStop = taskFlowTrainModel.getNextStop(nextStop);
  }
   //is either null or has the value of outcome
   return nextStopAction;
}


/**
 * Navigates to the previous stop in a train
 * @return outcome string
 */
public String navigatePreviousStop() {
```

```
  String prevStopAction = null;
      ControllerContext controllerContext =
                  ControllerContext.getInstance();
  ViewPortContext currentViewPortCtx =
                  controllerContext.getCurrentViewPort();
  TaskFlowContext taskFlowCtx =
                  currentViewPortCtx.getTaskFlowContext();
  TaskFlowTrainModel taskFlowTrainModel =
                  taskFlowCtx.getTaskFlowTrainModel();
  TaskFlowTrainStopModel currentStop =
                  taskFlowTrainModel.getCurrentStop();

  TaskFlowTrainStopModel prevStop =
                  taskFlowTrainModel.getPreviousStop(currentStop);
  while(prevStop != null){
    if(isSkipTrainStop(prevStop) == false){
       //no need to loop any further
       prevStopAction = prevStop.getOutcome();
       break;
     }
     prevStop = taskFlowTrainModel.getPreviousStop(prevStop);
   }

    //is either null or has the value of outcome
    return prevStopAction;
}


//We need check whether the next navigation stop should be skipped or
//not so the button navigation behaves the same as the train stop
//navigation. Otherwise skipped train stops would be navigated to
private boolean isSkipTrainStop(TaskFlowTrainStopModel stop){
  String activityId = stop.getLocalActivityId();
  //get access to the managed bean (HashMap) that keeps track of the
  //train stops that should be skipped
  FacesContext fctx = FacesContext.getCurrentInstance();
  ELContext elctx = fctx.getELContext();
  ExpressionFactory expressionFactory =
              fctx.getApplication().getExpressionFactory();

  ValueExpression ve = expressionFactory.createValueExpression(
                       elctx,
                       "#{pageFlowScope.skipHelperBean}",
                       Object.class);
  SkipHelperBean skipHelper = (SkipHelperBean) ve.getValue(elctx);
  Boolean skip = (Boolean) skipHelper.get(activityId);
```

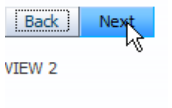
```
  return skip;
}

public void setFirstStop(boolean firstStop) {
  this.firstStop = firstStop;
}
//determine if no previous stop in train model
//return true if so
public boolean isFirstStop() {
  ControllerContext controllerContext =
                  ControllerContext.getInstance();
  ViewPortContext currentViewPortCtx =
                  controllerContext.getCurrentViewPort();
  TaskFlowContext taskFlowCtx =
                  currentViewPortCtx.getTaskFlowContext();
  TaskFlowTrainModel taskFlowTrainModel =
                  taskFlowCtx.getTaskFlowTrainModel();
  TaskFlowTrainStopModel currentStop =
                  taskFlowTrainModel.getCurrentStop();
  TaskFlowTrainStopModel prevStop =
                  taskFlowTrainModel.getPreviousStop(currentStop);
  if (prevStop == null){
    //only first stop has no previous stop
    return true;
  }
  else{
    return false;
  }
}

public void setLastStop(boolean lastStop) {
     this.lastStop = lastStop;
}
//determine if no further stops in train model
//return true if so
public boolean isLastStop() {
  ControllerContext controllerContext =
                  ControllerContext.getInstance();
  ViewPortContext currentViewPortCtx =
                  controllerContext.getCurrentViewPort();
  TaskFlowContext taskFlowCtx =
                  currentViewPortCtx.getTaskFlowContext();
  TaskFlowTrainModel taskFlowTrainModel =
```

```
                    taskFlowCtx.getTaskFlowTrainModel();
  TaskFlowTrainStopModel currentStop =
                    taskFlowTrainModel.getCurrentStop();
  TaskFlowTrainStopModel nextStop =
                    taskFlowTrainModel.getNextStop(currentStop);
  if (nextStop == null){
      //no next stop means that this is the last stop in the train
     return true;
  }
  else{
    return false;
  }
}
```

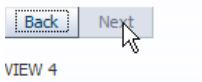
In the managed bean code above, four lines are highlighted in bold. The interesting with these lines is that they access the value of the train stop **Outcome** property. Usually you use this property to divert train navigation to include a method call activity or any activity else that can be accessed from a wild card control flow.

However, even when not setting this value, the train model returns an implicit outcome reference, which then is used on the command buttons to perform the train navigation. **This is the key** to programmatic navigation in ADF trains.

With this custom train button bar, as shown below, you can now skip a train stop that is disabled in the train model.



… step over. Notice in the image below that the **Next** button is disabled because it's the last stop in the train.



How-to navigate trains from component events

But how to navigate train models if not command item is involved? In this case, you can queue an action either on a hidden button, or if the task flow is exposed as an ADF region, on the region component. The latter is explained next as it can be coded more generic. The use case is that navigation should be to the next train stop in response to component events like `ValueChangeEvent`, `SelectioEvent`, and DVT `ClickEvent` or similar.

The way the code below works is that it gets the event origin, which the component (table, graph or inputTextField) that triggered the event. From here it then searches the parent component tree until it finds an instance of `RichRegion`, which is the instance of an `af:region` tag.

The managed bean code then determines the outcome of the next train stop, which then is queued on the region instance.

```
//queue the outcome of the next train stop
public void queueTrainNavigationNextAction(<SomeEvent> someEvent){
  UIComponent eventOriginComponent =
                              (UIComponent) actionEvent.getSource();
  RichRegion region = null;
  //find region
  UIComponent current = eventOriginComponent;
  while(current.getParent() != null){
    current = current.getParent();
    if (current instanceof RichRegion){
       region = (RichRegion) current;
        break;
    }
  }
  if (region != null) {
     ControllerContext controllerContext =
                      ControllerContext.getInstance();
     ViewPortContext currentViewPortCtx =
                      controllerContext.getCurrentViewPort();
     TaskFlowContext taskFlowCtx =
                      currentViewPortCtx.getTaskFlowContext();
     TaskFlowTrainModel taskFlowTrainModel =
                      taskFlowCtx.getTaskFlowTrainModel();
     TaskFlowTrainStopModel currentStop =
                      taskFlowTrainModel.getCurrentStop();
     TaskFlowTrainStopModel nextStop =
                      taskFlowTrainModel.getNextStop(currentStop);
     if (nextStop != null) {
         String nextStopActionStr = nextStop.getOutcome();
         //queue action
         region.queueActionEventInRegion(
                createMethodExpressionFromString(nextStopActionStr),
                null,null, false, new Integer(0), new Integer(0),
                PhaseId.INVOKE_APPLICATION);
      }
   }
}


//queue the outcome of the previous train stop
public void queueTrainNavigationPreviousAction(<SomeEvent> someEvent){
```

```
        UIComponent eventOriginComponent =
                        (UIComponent) actionEvent.getSource();
  RichRegion region = null;
  //find region
  UIComponent current = eventOriginComponent;
  while(current.getParent() != null){
    current = current.getParent();
    if (current instanceof RichRegion){
       region = (RichRegion) current;
        break;
    }
  }
  if (region != null) {
     ControllerContext controllerContext =
                        ControllerContext.getInstance();
     ViewPortContext currentViewPortCtx =
                        controllerContext.getCurrentViewPort();
     TaskFlowContext taskFlowCtx =
                        currentViewPortCtx.getTaskFlowContext();
     TaskFlowTrainModel taskFlowTrainModel =
                        taskFlowCtx.getTaskFlowTrainModel();
     TaskFlowTrainStopModel currentStop =
                        taskFlowTrainModel.getCurrentStop();
     TaskFlowTrainStopModel prevStop =
                        taskFlowTrainModel.getPreviousStop(currentStop);
     if (prevStop != null) {
        String prevStopActionStr = prevStop.getOutcome();
        //queue action
        region.queueActionEventInRegion(
                createMethodExpressionFromString(prevStopActionStr),
                null,null, false, new Integer(0), new Integer(0),
                PhaseId.INVOKE_APPLICATION);
       }
    }
 }


//create a method expression from a String
private MethodExpression createMethodExpressionFromString(String s){
  FacesContext fctx = FacesContext.getCurrentInstance();
  ELContext elctx = fctx.getELContext();
  ExpressionFactory exprFactory =
                        fctx.getApplication().getExpressionFactory();
  MethodExpression    methodExpr = exprFactory.createMethodExpression(
                                        elctx,
                                        s,
```

```
                                                    null,
                                                    new Class[]{});
  return methodExpr;
}
```

Conclusion

Train models can be navigated from Java, which is good to know as it allows you to implement your custom solutions for default component behaviors. This article helps you to customize the skip handling and explains how to navigate trains using Java.

The sample workspace can be downloaded as sample 82 from ADF Code Corner

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

Heads up: Oracle Magazine publishes a focus article about how to use trains in Oracle ADF applications in its September / October 2011 issue. The article provides a more comprehensive train sample, as well as additional insider hints and tips.

http://www.oracle.com/technetwork/oramag/magazine/home/index.html

RELATED DOCOMENTATION

⊠	Fusion Developer Guide – Trains http://download.oracle.com/docs/cd/E21764_01/web.1111/b31974/taskflows_complex.htm#CJHFBFIE
⊠	Page Templates in ADF http://download.oracle.com/docs/cd/E17904_01/web.1111/b31973/af_reuse.htm#CACCFCJC
⊠	Oracle Magazine http://www.oracle.com/technetwork/oramag/magazine/home/index.html