

ADF Code Corner

93. Put a different Look to your Train Stops



twitter.com/adfcodecorner

Abstract:

Creating sequential train models for navigation in bounded task flows is easy to achieve. To display a train model in a view, developers usually use the ADF Faces `af:train` or `af:trainButtonBar` component that show the train stops as iconic bullets on a horizontal line or as command buttons for next and previous navigation. To customize the default train rendering, developers could use skinning in ADF Faces, e.g. to change the icons used for the train stops.

However, with creativity – and if you are not shy of manual configuration – you can display trains with a custom layout component, like tabs. This article shows how to display train stop within a different look.

Author:

Frank Nimphius, Oracle Corporation
twitter.com/fnimphiu
13-DEC-2011

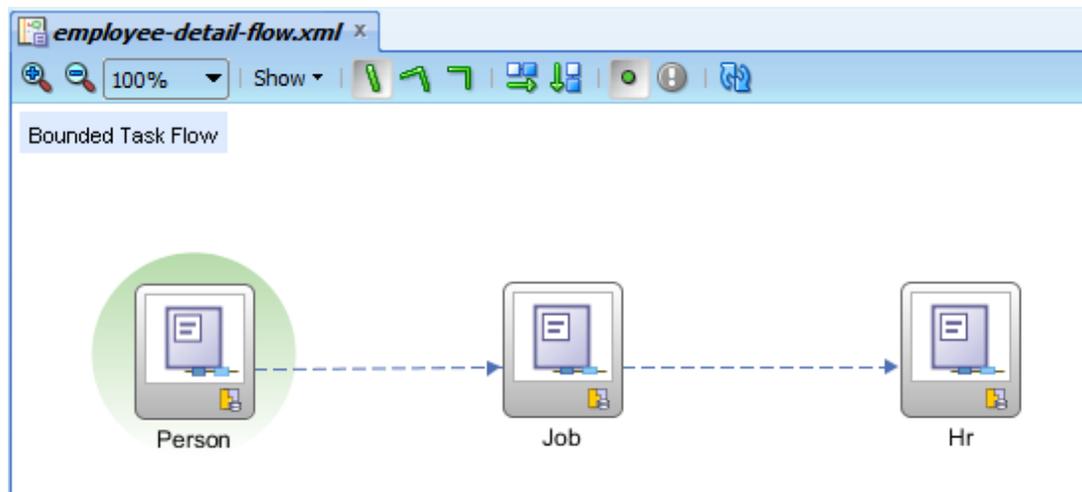
Oracle ADF Code Corner is a loose blog-style series of how-to documents that provide solutions to real world coding problems.

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

To create a train model for bounded task flows in ADF, you either select the train model option when creating the bounded task flow, or after the fact – set its **Train** property in the **Behavior** section of the **Property Inspector** to true. If then you enable view activities as a train stop, which you do by setting the **TrainStop** property of a view activity to **true**, the train model navigation shows in the task flow visual diagrammer as shown below.



Dragging an `af:train` or `af:trainButtonBar` component to a view automatically configures itself to the train model exposed on the ADF controller context object. At runtime the train stops show as in the image below if the `af:train` component is used,

DepartmentId	10		
DepartmentName	Administration		
ManagerId	200		
LocationId	1700		
<input type="button" value="First"/>	<input type="button" value="Previous"/>	<input type="button" value="Next"/>	<input type="button" value="Last"/>

EmployeeId	FirstName	LastName
201	Michael	Hartstein
202	Pat	Fay

Person Job Hr

FirstName

* LastName

* Email

With a few changes, which are in the focus of this blog article, you can change the default train rendering to a custom layout like shown below, in which each train stop is displayed as a tab.

DepartmentId	30		
DepartmentName	Purchasing		
ManagerId	114		
LocationId	1700		
<input type="button" value="First"/>	<input type="button" value="Previous"/>	<input type="button" value="Next"/>	<input type="button" value="Last"/>

EmployeeId
115
116
117
118
119

Person Job Hr

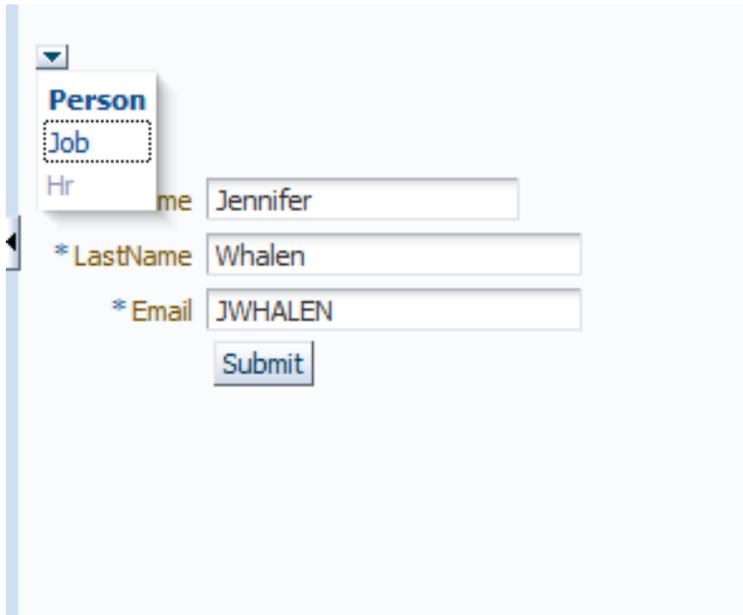
FirstName

* LastName

* Email

Note that using the tab layout, all the train features like sequential navigation and train stop skip behavior are reflected. It really is only the display of the stops that looks different – or shall I say great?

With a simple change in the configuration, the same train can render its stops as a choice ...



A screenshot of an ADF form. At the top left, there is a dropdown menu with a downward arrow. The menu is open, showing three options: 'Person' (highlighted in blue), 'Job' (highlighted with a dashed border), and 'Hr'. Below the dropdown, the form contains three text input fields: 'Name' with the value 'Jennifer', '* LastName' with the value 'Whalen', and '* Email' with the value 'JWHALEN'. A 'Submit' button is located below the email field.

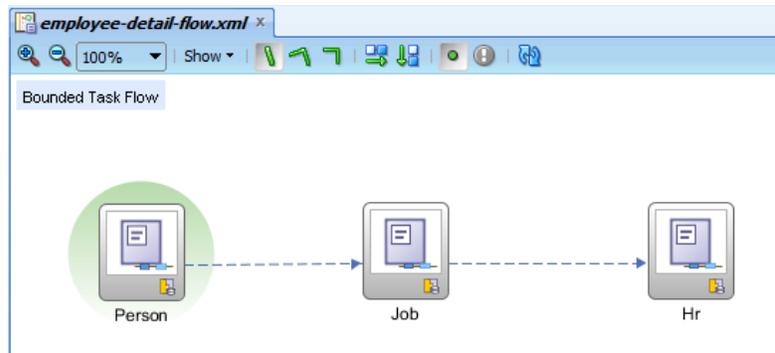
... or, a button bar



A screenshot of an ADF form. At the top left, there is a button bar with three buttons: 'Person' (highlighted in blue), 'Job', and 'Hr'. Below the button bar, the form contains three text input fields: 'FirstName' with the value 'Jennifer', '* LastName' with the value 'Whalen', and '* Email' with the value 'JWHALEN'. A 'Submit' button is located below the email field.

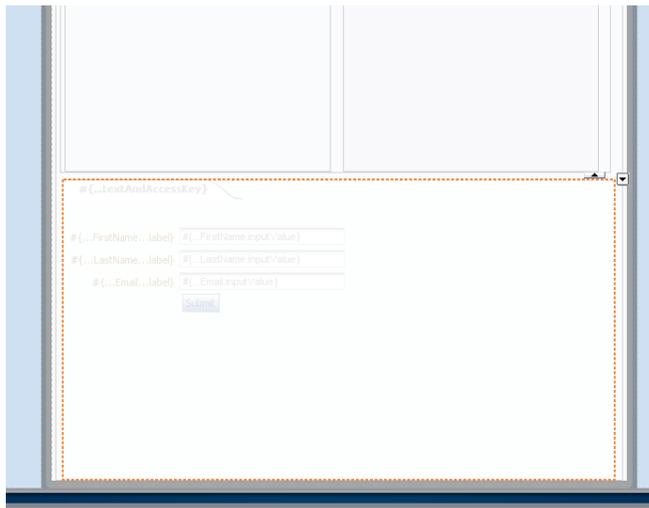
The Bounded Task Flow

Train models can be automatically created for bounded task flows. So the pre-requisite put on this article is that the task flow you work with is bounded.



Train models can be displayed for bounded task flows that use stand-alone pages (JSPX documents or Facelets) and task flows executing as part of a page or view (ADF region). A task flow that is displayed in a region launched by the Dynamic Tab Shell template for example, may look much better if the train model stops are rendered using tabs.

The sample workspace that you can download at the end of this article exposes the bounded task flows in an ADF region added to a page.



Note: The views in the sample bounded task flow are all based on the same view object row, Employee, to simulate a form entry wizard. Because all forms on the views "speak" to the same row in the Data Control it is necessary to defer validation until the end.

For this, on the **PageDef** files associated with the views in the bounded task flow in this sample, the **SkipValidation** property is set to **true** to suppress validation until when validation should be enforced. Because the ADF binding layer does not support partial submit of Data Controls, you need to defer validation this way until all form fields are entered.

Building the custom train components

To display the train stop navigation exposed by the bounded Task Flow train model, the sample uses the ADF Faces `af:navigationPane` component. The `af:navigationPane` component supports the Trinidad `MenuModel`, which is also implemented by the Train Model.

Using the **hint** property of the `af:navigationPane` component, the rendering of the train stop display can be configured as tabs, list, choice and button bar as shown in the image below.

```
<af:navigationPane hint="" value="#{controllerContext.currentViewPort.taskFlowContext.trainModel}"
var="trainNode" id="np1">
  <f:facet name="nodeStamp">
    <af:commandNavigationItem text="#{trainNode.textAndAccessKey}" id="cni1"
visited="#{trainNode.visited}" disabled="#{trainNode.disabled}"
action="#{trainNode.action}" selected="#{trainStopBean.currentTab}"/>
  </f:facet>
</af:navigationPane>
```

The page source of the `af:navigationPane` component is added to the page fragments in the bounded task flow, which also could be done as part of a page template.

```
<af:navigationPane hint="tabs"
value="#{controllerContext.currentViewPort.taskFlowContext.trainModel}"
var="trainNode" id="np1">
  <f:facet name="nodeStamp">
    <af:commandNavigationItem text="#{trainNode.textAndAccessKey}" id="cni1"
visited="#{trainNode.visited}" disabled="#{trainNode.disabled}"
action="#{trainNode.action}" selected="#{trainStopBean.currentTab}"/>
  </f:facet>
</af:navigationPane>
```

As shown in the page source above, all command item settings are directly read from the train model, which for each stop populates the **trainNode** temporary variable upon train rendering. The only setting that could not be read directly from the train model, which however is required when the train stops are rendered as tabs, is the **selected** property. For this, the sample uses a managed bean in none scope (as it does not keep any state).

The managed bean is configured in the metadata configuration file of the bounded task flow so when the task flow is deployed separately, the managed bean configuration is not getting lost

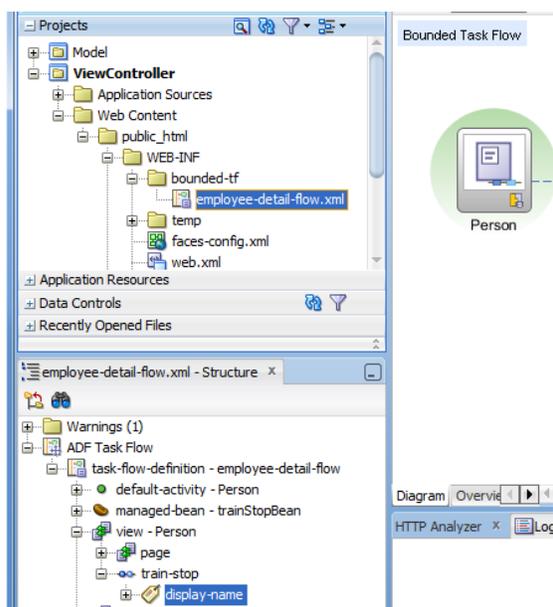
```
public class TrainStopManagedBean {
    public TrainStopManagedBean() {
        super();
    }
    public boolean isCurrentTab() {
        //get access to the JSF context classes
        FacesContext fctx = FacesContext.getCurrentInstance();
        ELContext elctx = fctx.getELContext();
        Application app = fctx.getApplication();
        ExpressionFactory expressionFactory = app.getExpressionFactory();
        //trainNode is the name of the variable attribute defined in
        //af:navigationPane
```

```

ValueExpression ve = expressionFactory.createValueExpression
    (elctx,"#{trainNode}", Object.class);
//get the rendered stop's viewActivity
TaskFlowTrainStopModel renderedTrainNode =
    (TaskFlowTrainStopModel)ve.getValue(elctx);
//get current train stop to compare it with the current "rendered"
//train stop
ControllerContext controllerContext =
    ControllerContext.getInstance();
ViewPortContext currentViewPortCtx =
    controllerContext.getCurrentViewPort();
TaskFlowContext taskFlowCtx =
    currentViewPortCtx.getTaskFlowContext();
TaskFlowTrainModel taskFlowTrainModel =
    taskFlowCtx.getTaskFlowTrainModel();
//the train stop that is rendered in the train bar
String renderedActivityId = renderedTrainNode.getLocalActivityId();
//the train's current stop: the state
TaskFlowTrainStopModel currentStop =
    taskFlowTrainModel.getCurrentStop();
if (renderedActivityId.equalsIgnoreCase(
    currentStop.getLocalActivityId())) {
    return true;
}
return false;
}
}

```

Note: To define the train stop label, you need to use the Structure Window in Oracle JDeveloper as shown below



To add labels to your train stops. Select the view activity and open the Structure Window. Select the activity -> train stop node with the right mouse button and choose **Insert Inside train-stop** from the menu, Choose **Display Name** and from the sub-menu and provide a label.

Sample Download

The sample workspace is for Oracle JDeveloper 11.1.2.1. The solution however works with 11.1.1.x released of Oracle JDeveloper 11g as well. You can download the workspace as sample 93 from

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

The database connections needs to be configured to point to a local HR schema of an Oracle XE, standard or enterprise database

RELATED DOCUMENTATION

<input type="checkbox"/>	Navigation Pane http://docs.oracle.com/cd/E21764_01/apirefs.1111/e12419/tagdoc/af_navigationPane.html
<input type="checkbox"/>	Oracle Magazine: Trains "All Aboard" http://www.oracle.com/technetwork/issue-archive/2011/11-sep/o51adf-452576.html
<input type="checkbox"/>	Programmatically navigating trains http://www.oracle.com/technetwork/issue-archive/2011/11-sep/o51adf-452576.html