ADF Code Corner

80. HashMap strategy for dynamically setting the sequential property in ADF Controller train models

Abstract:

ADF Controller bounded task flows can be configured to generate a train model at runtime. The train model is then referenced from an `af:train` or `af:trainButtonBar` component added to the views in the bounded task flow to navigate within multi step processes. Using the train, each step becomes a train stop that can be configured to be accessible sequentially or non.sequentially (which means random) by the user. The configuration for the type of sequential access is defined individually on the view and task flow call activity. Expression language can be used to dynamically set the train stop sequential property.

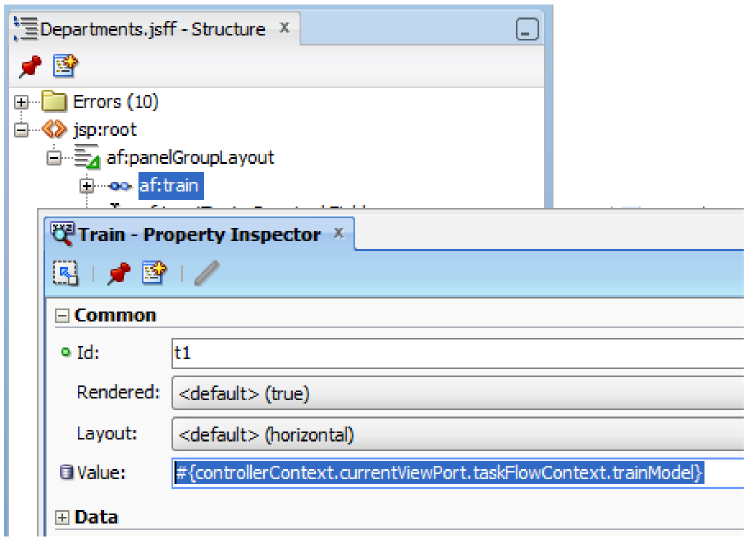
This article explains a strategy that uses a HashMap as a managed bean to allow declarative configuration at design time and dynamic changes at runtime using Java and EL.

twitter.com/adfcodecorner

Author: Frank   Nimphius, Oracle Corporation
twitter.com/fnimphiu
30-APR-2010

Introduction

View activities and task flow call activities can be configured to become train stops in a train model that is referenced from `af:train` or af:`trainButtonBar` components in the pages.



Id: t1
Rendered: <default> (true)
Layout: <default> (horizontal)
Value: #{controllerContext.currentViewPort.taskFlowContext.trainModel}

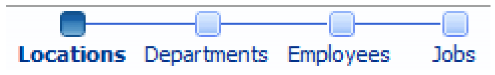
By default, train stops are sequential, which means that stops are enabled based on their position relative to the current stop. Only visited stops and the one immediately following the current stop are enabled. Through configuration, individual stops can be changed to non-sequential access to allow users to skip train stops between the current and the non-sequential stop.

Using expression language (EL), it is possible to dynamically set the sequential property of a stop, which allows you to dynamically determine whether stops must be visited or if users are allowed to skip them. Note that the use case of users skipping a train stop is not the same as developers configuring the *skip* or *ignore* properties on a property. The skip and ignore property either suppress (diable) a train stop navigation or hode the stop alltogether. Dyanamically setting the *sequential* property however makes navigation optional for the user, who still may decide to continue with the sequence of stops.

This ADF Code corner article explains how to use configure a HashMap as a managed bean for declarative and dynamic configuration of the train stop sequential.property. Using the same approach you can create configurable metadata for other properties, for example to show train stops as required upon rendering or defining the skip property,

About trains in Oracle ADF

Trains in Oracle ADF Controller are navigation links within views of a bounded task flows that allow users to see their current position in a multi step process and to navigate between views. Bounded task flows can be configured to become train models by setting of a single property.

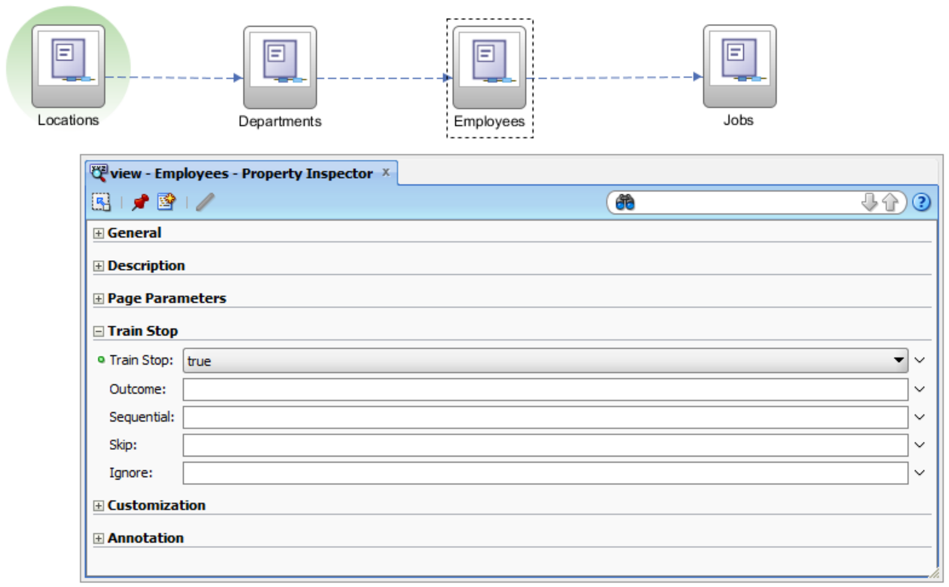


Within each bounded task flow, view activities and task flow call activities can be defined to become train stops. The train stop behavior is configured through properties

- *skip* - to disable a stop so user navigation steps over it
- *sequential* – to define navigation to be sequential for a specific stop or not
- *ignore* – to hide a train stop upon task flow initialization (not later though)
- *outcome* – allows defining a control flow case that the train follows when the train stop is clicked. This allows you to navigate e.g. to a method call activity first before accessing a view

Read more about trains:
http://download.oracle.com/docs/cd/E17904_01/web.1111/b31974/taskflows_complex.htm#CJHFBFIE

Using a HashMap to configure the sequential property



Train Stop: true
Outcome:
Sequential:
Skip:
Ignore:

When opening the task flow diagram for a bounded task flow, views that are marked as **Train Stops** are connected by a dotted line. Though not shown in the image above, using a train for navigating bounded task flows does not exclude the use of control flow cases for navigation. Control flow cases could be used for programmatic navigation.

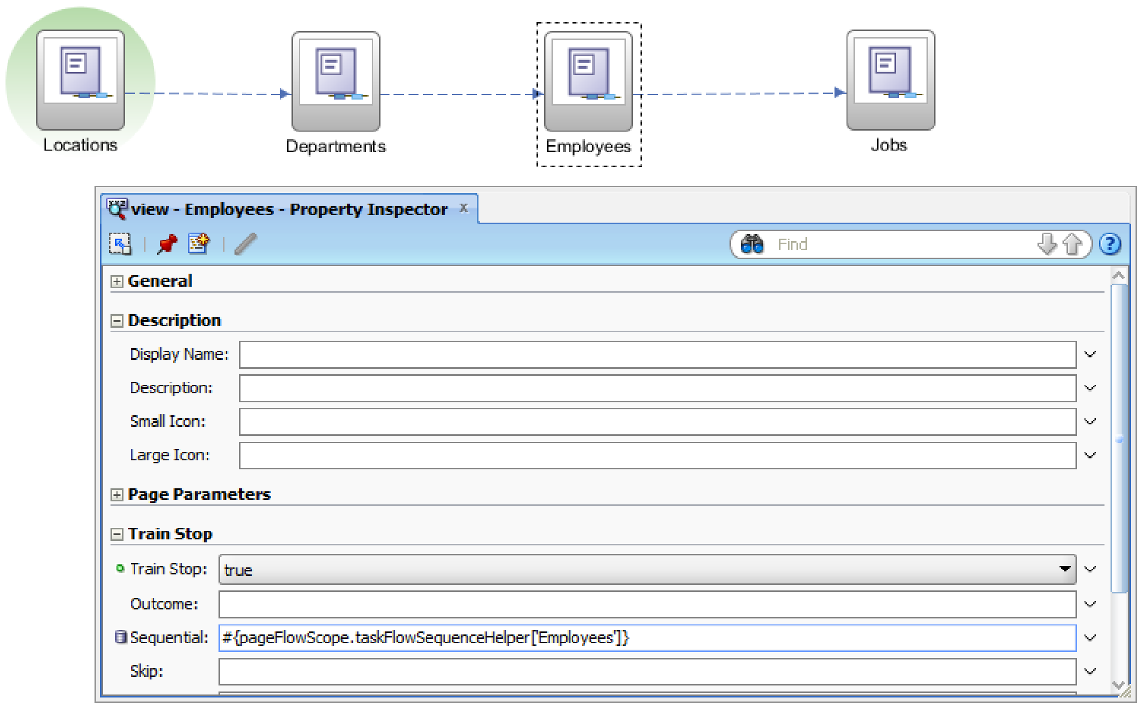
As shown in the image above, the Property Inspector provides you with a choice of configuration settings that you can use to define the characteristic and behavior of a train stop.

However, instead of typing **true** or **false** into the **Sequential** property field, in this article, I explain how to use a more dynamic configuration that becomes handy if you need to change the sequential behavior based on a current condition.

Note: while the *Skip* and *Sequential* properties can be changed at task flow runtime, *ignore* can only be changed upon task flow initialization, not later. So to hide a train stop at runtime, you would need to restart the complete task flow, which for some use cases may be an option (e.g. on an airport, you provide the information whether you check in luggage or not before actually starting the checking process. So the luggage check-in stop will be hidden. If you then change your mind, you just re-start the check-in process.

The image below shows the changed configuration for the **Sequential** property. Note that the value now is an EL string pointing to a managed bean in **pageFlowScope**.

The view activity Id is passed to the managed bean as an argument so the Boolean value **true** or **false** can be determined based on this. But how can an argument be passed to a managed bean method? Well, this is because the managed bean is an extension of HashMap.



Display Name:	
Description:	
Small Icon:	
Large Icon:	
Train Stop:	true
Outcome:	
Sequential:	#{pageFlowScope.taskFlowSequenceHelper['Employees']}
Skip:	

Creating the managed bean

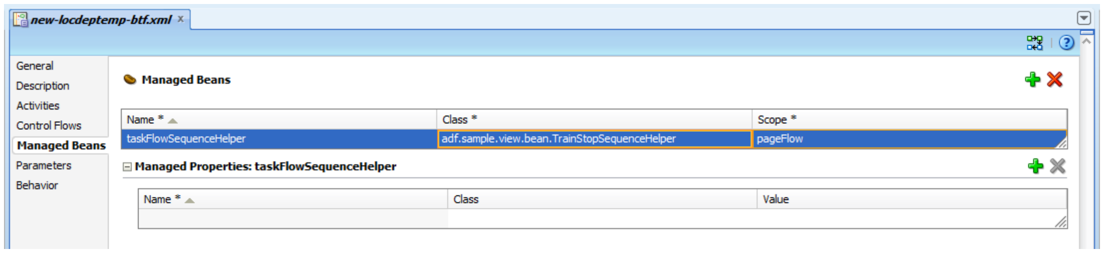
The most important methods of a HashMap – at least for this use case – are its **get** method and its **put** method. The **get** method takes a single argument as a key to identify a stored value. The **put** method allows to add a new object identified by a key argument or to replace an existing entry.

For the use case of configuring the **Sequential** train-stop property, we need to override the **get** method to return a Boolean type. When configuring the managed bean, all views and their sequential behavior are defined as strings. For this to work, we need a location that safely translates strings back into their boolean representation.

```
import java.util.HashMap;

public class TrainStopSequenceHelper extends HashMap {

 @SuppressWarnings("compatibility:-5522658446910108463")
  private static final long serialVersionUID = 7511234320471184490L;


  public TrainStopSequenceHelper() {
        super();
  }

  //returns sequenctial setting for view activity identified by the
  //key value. Returns true if activity is not recognized (which can be
  //due to misspelling or missing configuration in the bounded task
  //flow definition) or when the stop is configured for sequential
  //access.
  @Override
    public Object get(Object key) {
        if(super.get(key) ==null){
          return Boolean.TRUE;
        }
        try {
            return Boolean.valueOf((String)super.get(key));
        } catch (Exception e) {
            //if value cannot be converted into boolean then
            //return true to define a sequential stop.
            return true;
        }
    }
}
```

Configuring the managed bean

The managed bean is configured in the task flow configuration '.xml' file as shown below



Managed Beans

Name *	Class *	Scope *
taskFlowSequenceHelper	adf.sample.view.bean.TrainStopSequenceHelper	pageFlow

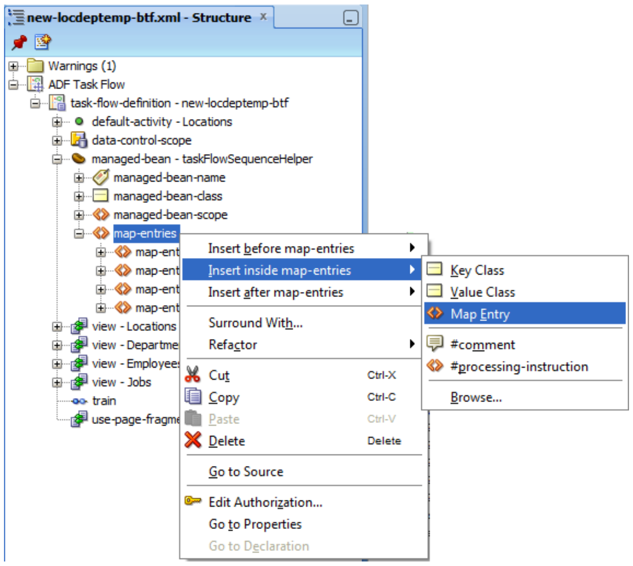
Managed Properties: taskFlowSequenceHelper

Name *	Class	Value

While this looks like an ordinary managed bean configuration in a task flow, it is not. The task flow overview dialog does not show map entries, which are specific configurations for HashMap beans to "inject" default values.

```
<managed-bean id="__15">
  <managed-bean-name id="__13">
    taskFlowSequenceHelper
  </managed-bean-name>
  <managed-bean-class id="__11">
    adf.sample.view.bean.TrainStopSequenceHelper
  </managed-bean-class>
  <managed-bean-scope id="__14">
    pageFlow
  </managed-bean-scope>
  <map-entries>
    <map-entry id="__30">
      <key id="__31">Locations</key>
      <value id="__32">true</value>
    </map-entry>
    <map-entry id="__33">
      <key id="__34">Departments</key>
      <value id="__35">false</value>
    </map-entry>
    <map-entry id="__19">
      <key id="__18">Employees</key>
      <value id="__17">false</value>
    </map-entry>
    <map-entry id="__36">
      <key id="__37">Jobs</key>
      <value id="__38">false</value>
    </map-entry>
  </map-entries>
</managed-bean>
```

Shown in the image below, using the Oracle JDeveloper Structure Window, you can create the map entries for the HashMap bean. The map entries are passed to the managed bean upon bean instantiation and define the default configuration for the **Sequential** property of the view activities defined as train stops.

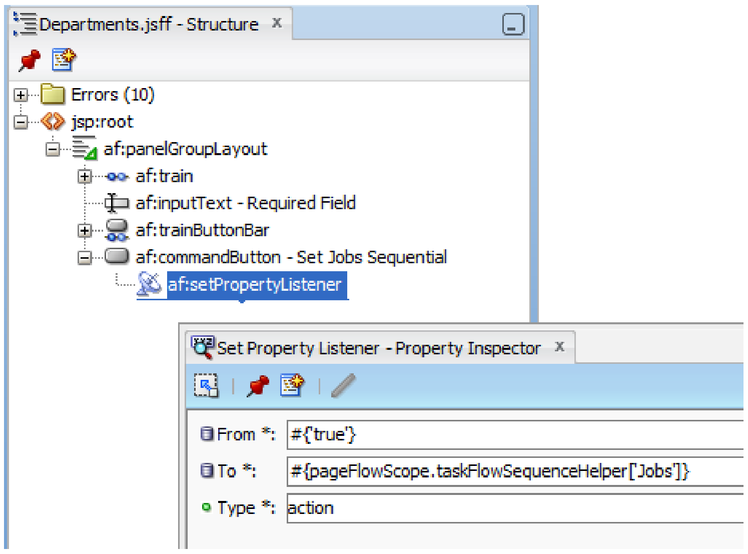


At runtime, you can change the **Sequential** settings from **Java** by accessing the managed bean from a Value Expression

```
FacesContext fctx = FacesContext.getCurrentInstance();
ELContext elctx = fctx.getELContext();
ExpressionFactory expressionFactory =
          fctx.getApplication().getExpressionFactory();
ValueExpression ve = null;
ve = expressionFactory.createValueExpression(
                    elctx,


                    "#{pageFlowScope.taskFlowSequenceHelper }",
                    Object.class);
//get the rendered stop's viewActivity
TrainStopSequenceHelper  sequenceHelper = null;
sequenceHelper = (TrainStopSequenceHelper) ve.getValue(elctx);
sequenceHelper.put("Employees","false");
```

Or you use the `af:setPropertyListener` on a command item like `af:commandButton` and set the value declaratively

```
<af:commandButton text="Set Jobs Sequential" id="cb1">
    <af:setPropertyListener from="#{'true'}"
                            to="#{pageFlowScope.taskFlowSequenceHelper['Jobs']}"
                            type="action"/>
 </af:commandButton>
```


Departments.jsff - Structure
Errors (10)
jsp:root
af:panelGroupLayout
af:train
af:inputText - Required Field
af:trainButtonBar
af:commandButton - Set Jobs Sequential
af:setPropertyListener
Set Property Listener - Property Inspector

From *: #{'true'}
To *: #{pageFlowScope.taskFlowSequenceHelper['Jobs']}
Type *: action

Note that in both cases, using Java or `af:setPropertyListener`, the train component needs to be refreshed either through navigation to another view, or PPR.

Conclusion

Using extended HashMaps as managed beans greatly simplify the dynamic configuration of train stop behavior. You can also use task flow input parameters to set values for the keys stored in the HashMap, allowing you to override the default configuration done during development time.

Note: This article does not provide a sample to download

RELATED DOCOMENTATION

☒	ADF Faces train component tag http://download.oracle.com/docs/cd/E17904_01/apirefs.1111/e12419/tagdoc/af_train.html
☒	ADF Faces train button bar component http://download.oracle.com/docs/cd/E17904_01/apirefs.1111/e12419/tagdoc/af_trainButtonBar.html
☒	Creating trains – product documentation http://download.oracle.com/docs/cd/E17904_01/web.1111/b31974/taskflows_complex.htm#CJHFBFIE