ADF Code Corner
Oracle JDeveloper OTN Harvest 11 / 2011

twitter.com/adfcodecorner

Abstract:

The Oracle JDeveloper forum is in the Top 5 of the most active forums on the Oracle Technology Network (OTN). The number of questions and answers published on the forum is steadily increasing with the growing interest in and adoption of the Oracle Application Development Framework (ADF).

The ADF Code Corner "Oracle JDeveloper OTN Harvest" series is a monthly summary of selected topics posted on the OTN Oracle JDeveloper forum. It is an effort to turn knowledge exchange into an interesting read for developers who enjoy harvesting little nuggets of wisdom.

http://blogs.oracle.com/jdevotnharvest/

Author: Frank Nimphius, Oracle Corporation
twitter.com/fnimphiu
30-NOV-2011

November 2011 Issue – Table of Contents

Using Groovy in Entity Validation

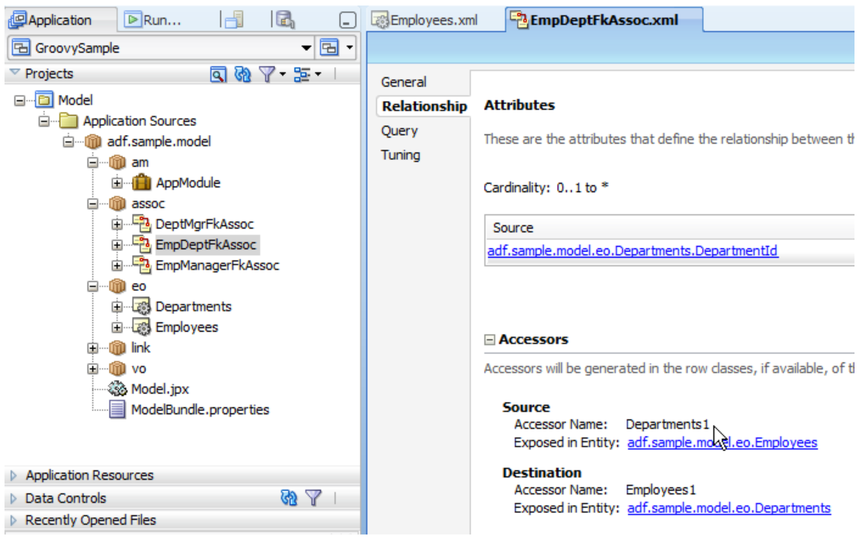
The use case for this sample has been posted on OTN as follows: "How to define entity validation for the **Salary** attribute of the Employees table so that pay raises going beyond the cumulated salary budget for a departments throws an exception". The logic for this is as follows:

AvailableBudget – (AllSalariesInDepartment + newEmployeeSalar) + currentEmployeeSalary < 0

The available budget is compared to all salaries paid in a department. Chances are that some salaries have been changed for an entity but not yet committed, which means thatyou need to read the salary from the entity cache to obtain the current budget. You then add the old employee salary before subtracted the new salary from the available budget to determine if the pay raise is within budget.

To implement the solution in Groovy, I assume you have an ADF Business Component model built on the Departments and Employees table of the Oracle HR Schema. You should have two entities created with an association defined between.

Select the **association** and double click on it to show the visual editor. Click onto the **Relationship** menu item to see the **Source** and **Destination** accessor names (e.g. Departments1, Employees1). Using accessors, you can "walk" the entity relationship up and down (e.g. to access the Departments entity from the Employees entity), which is key to this little recipe.



Attributes

These are the attributes that define the relationship between th

Cardinality: 0..1 to *

Source
adf.sample.model.eo.Departments.DepartmentId

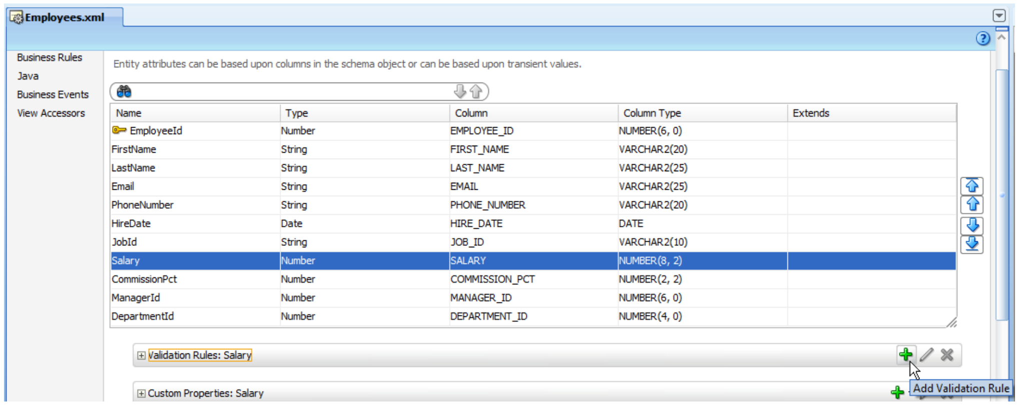
Accessors

Accessors will be generated in the row classes, if available, of th

Source
Accessor Name: Departments1
Exposed in Entity: adf.sample.model.eo.Employees

Destination
Accessor Name: Employees1
Exposed in Entity: adf.sample.model.eo.Departments

In the Employee entity editor, select the **Salary** attribute and click the **green plus** icon next to the **Validation** section to create a new validation rule.



Employees.xml

Business Rules
Java
Business Events
View Accessors

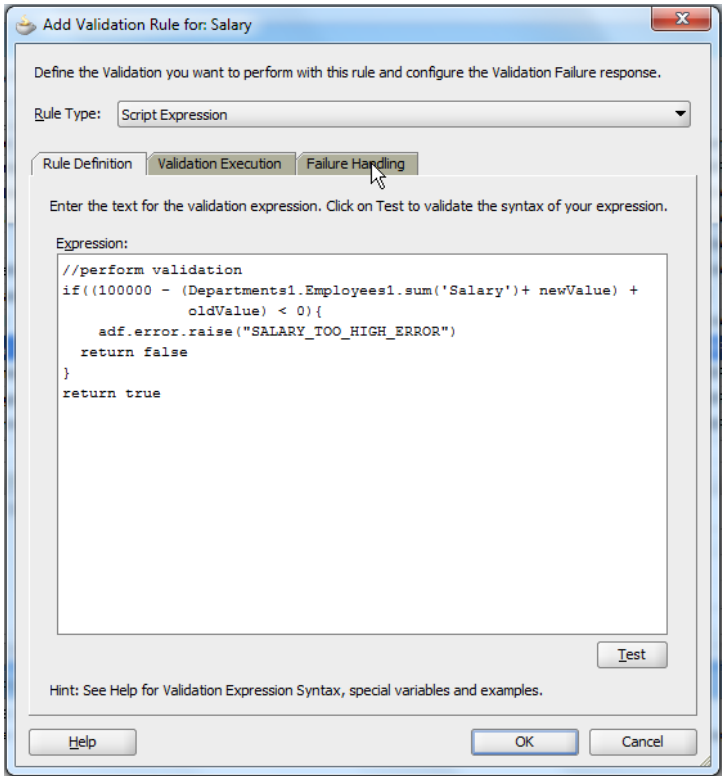
Entity attributes can be based upon columns in the schema object or can be based upon transient values.

Name	Type	Column	Column Type	Extends
EmployeeId	Number	EMPLOYEE_ID	NUMBER(6, 0)	
FirstName	String	FIRST_NAME	VARCHAR2(20)	
LastName	String	LAST_NAME	VARCHAR2(25)	
Email	String	EMAIL	VARCHAR2(25)	
PhoneNumber	String	PHONE_NUMBER	VARCHAR2(20)	
HireDate	Date	HIRE_DATE	DATE	
JobId	String	JOB_ID	VARCHAR2(10)	
Salary	Number	SALARY	NUMBER(8, 2)	
CommissionPct	Number	COMMISSION_PCT	NUMBER(2, 2)	
ManagerId	Number	MANAGER_ID	NUMBER(6, 0)	
DepartmentId	Number	DEPARTMENT_ID	NUMBER(4, 0)	

Validation Rules: Salary

Custom Properties: Salary

In the **Add Validation Rule for Salary** dialog, choose the **Rule Type** as **Script Expression**



Add Validation Rule for: Salary

Define the Validation you want to perform with this rule and configure the Validation Failure response.

Rule Type: Script Expression

Enter the text for the validation expression. Click on Test to validate the syntax of your expression.

Expression:

```
//perform validation
if((100000 - (Departments1.Employees1.sum('Salary')+ newValue) +
              oldValue) < 0){
    adf.error.raise("SALARY_TOO_HIGH_ERROR")
  return false
}
return true
```

Hint: See Help for Validation Expression Syntax, special variables and examples.

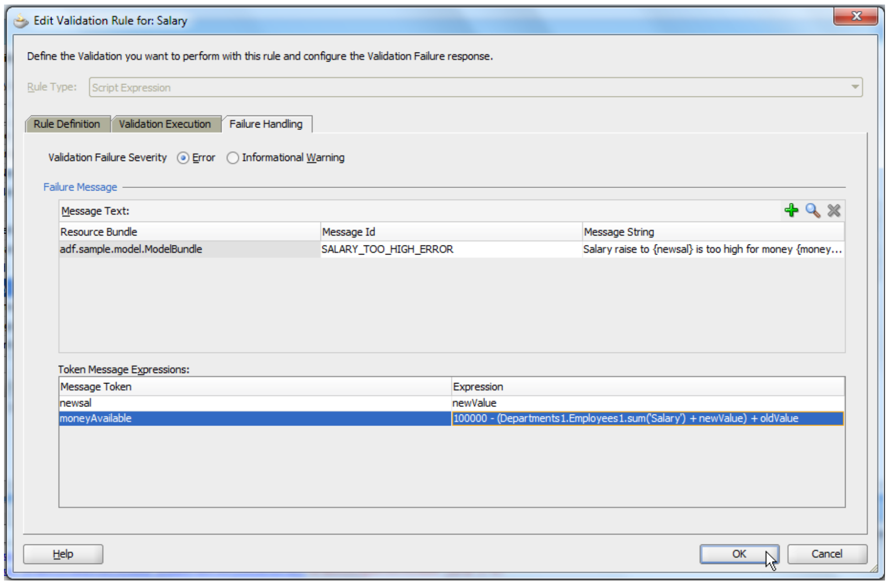
As the Rule Definition, add the following script:

```
//perform validation
if((100000 - (Departments1.Employees1.sum('Salary')+ newValue) +
              oldValue) < 0){
  adf.error.raise("SALARY_TOO_HIGH_ERROR")
  return false
}
return true
```

As you can see from the script, it uses the entity association accessors to get to the **Departments** entity to then compute the summary of salaries paid within.

What you need to know: The validation provides implicit object to access information contained within. Two of these information is **oldValue** and **newValue**, which gives you a handle to the value as it was before (committed or uncommitted) and the new value the attribute is changed to. Another useful object, though not used here is **source**. The **source** object gives you access to the entity object itself. You can use this e.g. to access public methods in an entity impl class (no need to expose them on the client interface). A use case for this is if you wanted to access **getPostedData** which is the data originally queried from the database. The **getPostedData** is a protected method, which is why you need a custom entity impl class to expose it through a public method, just the need to use the **source** object.

Select the **Failure Handling** tab of the dialog to define the message to be shown when validation fails. Add message string with variables. Variables are defined by curly braces and a name, like {newsal}


Edit Validation Rule for: Salary

Define the Validation you want to perform with this rule and configure the Validation Failure response.

Rule Type: Script Expression

Rule Definition | Validation Execution | Failure Handling

Validation Failure Severity [✓ Error] Informational Warning

Failure Message

Message Text:

Resource Bundle	Message Id	Message String
adf.sample.model.ModelBundle	SALARY_TOO_HIGH_ERROR	Salary raise to {newsal} is too high for money {money...

Token Message Expressions:

Message Token	Expression
newsal	newValue
moneyAvailable	100000 - (Departments1.Employees1.sum('Salary') + newValue) + oldValue

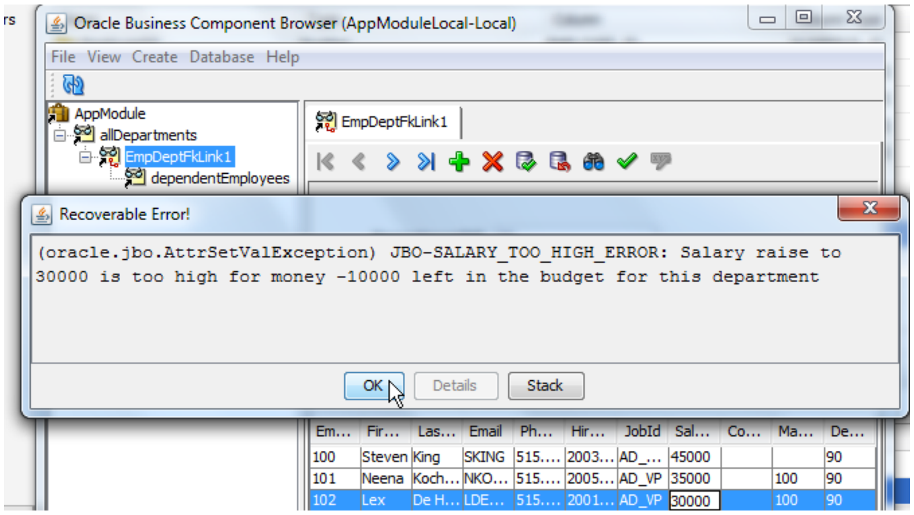
The message is added to a resource properties file. In the sample I used a message key of: SALARY_TOO_HIGH_ERROR. The message string associated to this in the properties file is shown below:

SALARY_TOO_HIGH_ERROR=Salary raise to **{newsal}** is too high for money **{moneyAvailable}** left in the budget for this department

The variable names can then be configured in the **Failure Handling** tab, e.g. to reference the new value and the budget.

You then run the AM and add salaries so that the budget is existed and the message will show.

Using Groovy for this use case is straight forward though, as I admit, requires some better knowledge of ADF Business Components and Java.



```
(oracle.jbo.AttrSetValException) JBO-SALARY_TOO_HIGH_ERROR: Salary raise to
30000 is too high for money -10000 left in the budget for this department
```

Em...	Fir...	Las...	Email	Ph...	Hir...	JobId	Sal...	Co...	Ma...	De...
100	Steven	King	SKING	515....	2003...	AD_...	45000			90
101	Neena	Koch...	NKO...	515....	2005...	AD_VP	35000		100	90
102	Lex	De H...	LDE...	515....	2001...	AD_VP	30000		100	90

Before you ask: To make the budget generic, you can have an attribute added to the Departments table that holds the value, in which case it too becomes Groovy accessible

Using ADF logger with Groovy in ADF BC

Groovy is a scripting language that can be used in the context of ADF BC. It simplifies Java object access and method execution. Basically everything you could access from Java you can access from Groovy. If logging is needed in your Groovy scripting, then this can be achieved using the ADF Logger as shown below. Just add script similar to this into the Groovy code are (e..g when working with custom entity validators)

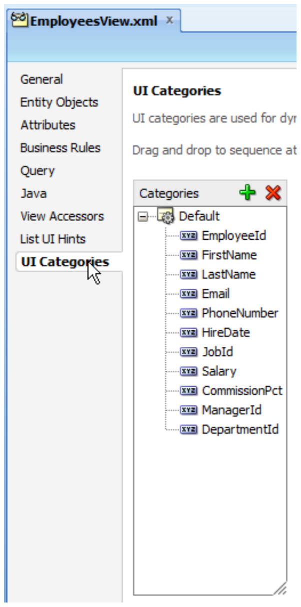
```
//log information to console. Create a logger instance for
//the entity class - "Employees in this sample

oracle.adf.share.logging.ADFLogger LOGGER =
          oracle.adf.share.logging.ADFLogger.
          createADFLogger(source.getClass())

LOGGER.info("Groovy Log Statement: oldValue and newValue in Validdator
            "+oldValue+" ---> "+newValue)

LOGGER.info("Groovy Log Statement: All salaries in Department
            "+Departments1.Employees1.sum('Salary'))

LOGGER.info("Groovy Log Statement: current budget "+(100000 -
        (Departments1.Employees1.sum('Salary')+ newValue) + oldValue ))
```

Like In Java, you define a variable to hold the logger reference to then use it to print info messages

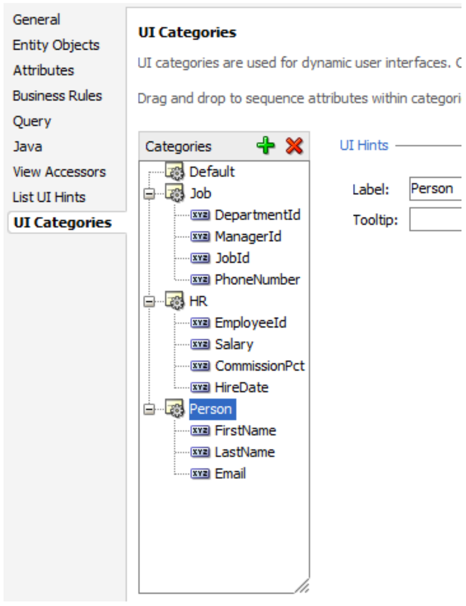
UI Categories in ADF BC of JDeveloper 11g R2

Oracle ADF Business Components in Oracle JDeveloper 112g R2 allow to categorize attributes in a View Object so they can be displayed in UI groups in the generated UI of the ADF BC tester and – with a bit of configuration – in the ADF Faces view.

To use categorization, **open** the **View Object** from the Application Navigator.

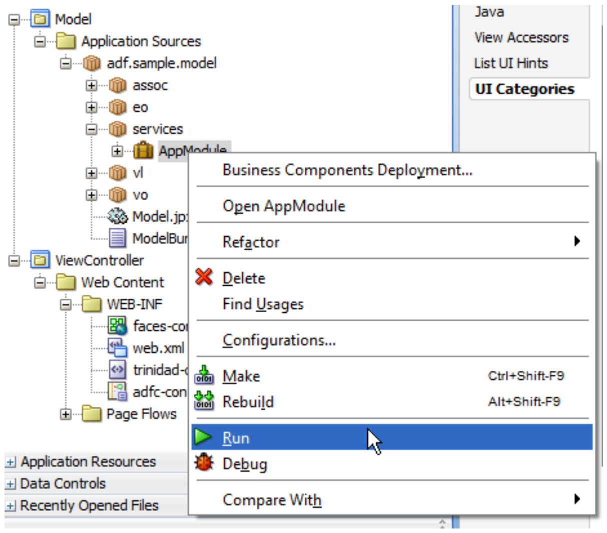


A **default** category exists that yet holds all attributes. Use the **green plus icon** to create additional categories.

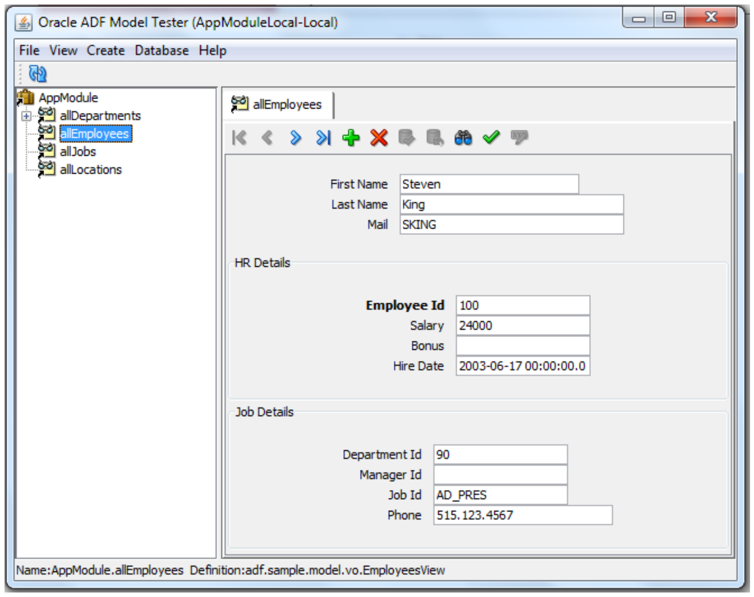


Note: You cannot order the categories in the ADF Business Component Editor.

Create additional categories as needed (like Job and HR in this example) and drag the attribute into the categories.



Run the Application Module in the ADF Business Component tester to see the categories in action.


Oracle ADF Model Tester (AppModuleLocal-Local)
File View Create Database Help
AppModule
allDepartments
allEmployees
allJobs
allLocations
allEmployees

First Name: Steven
Last Name: King
Mail: SKING

HR Details

Employee Id: 100
Salary: 24000
Bonus:
Hire Date: 2003-06-17 00:00:00.0

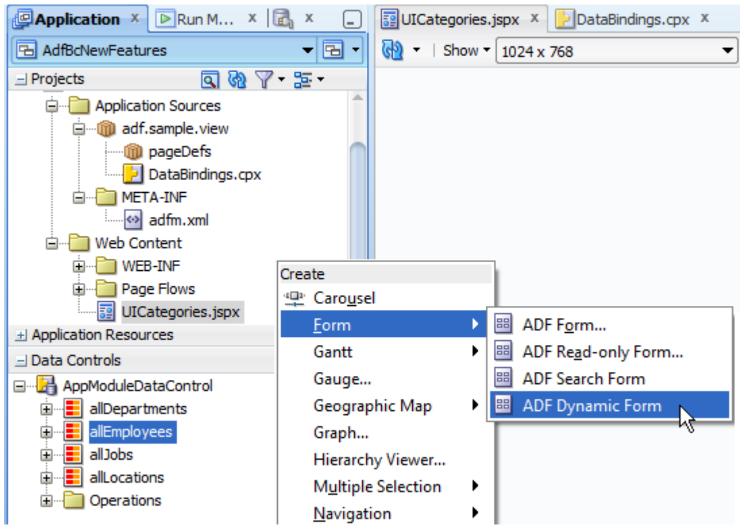
Job Details

Department Id: 90
Manager Id:
Job Id: AD_PRES
Phone: 515.123.4567

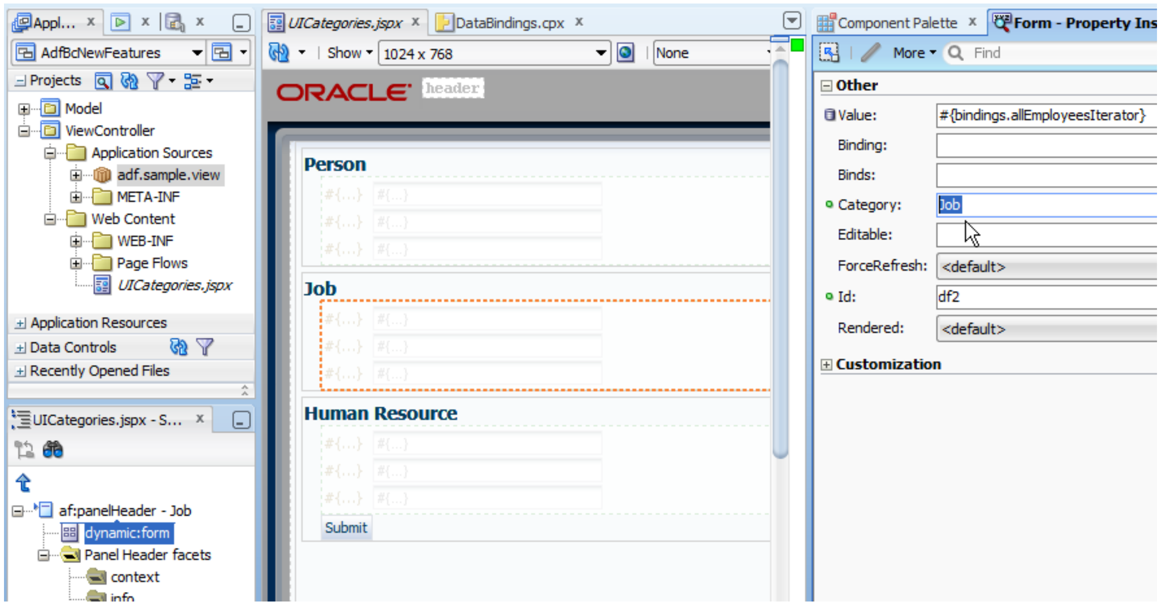
Name:AppModule.allEmployees Definition:adf.sample.model.vo.EmployeesView

To use ADF Business Components categories on ADF Faces pages, create a **JSPX document** and drag the View Object collection as an **ADF Dynamic Form**

Note: ADF Dynamic Forms are not yet supported with Facelets in Oracle JDeveloper 11.1.2.1



Create as many ADF Dynamic Forms as you have categories. In the example there are three categories: Person, Job, and HR. Select one of the `dynamic:form` tags and open the Properties Inspector. Set the **Category** Property to the name of the property to render.



At runtime, the categories attributes are automatically added to the dynamic form representing a category. Note that the above layout is achieved by surrounding the `dynamic:form` withan `af:panelHeader` component.

To dynamically set the af:panelHeader title to the UI hint label specified for a category, use Expression Language as shown below

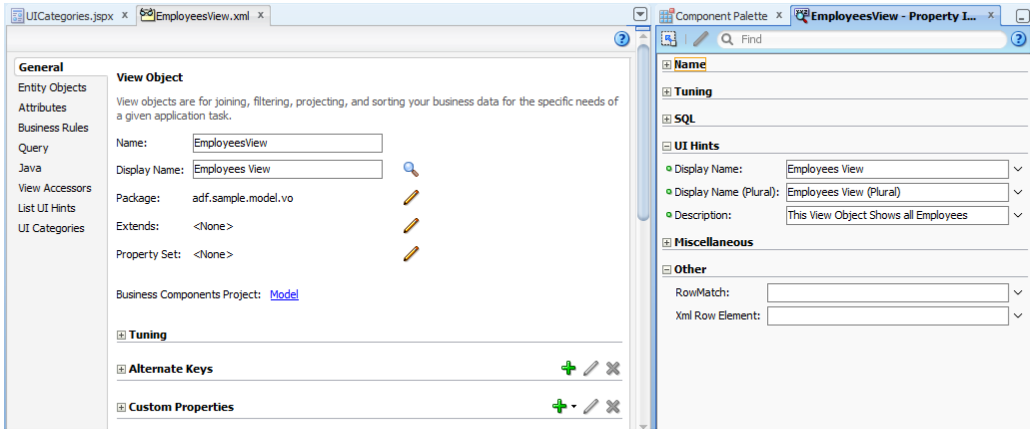
```
<af:panelGroupLayout id="pgl1" layout="vertical">
  <af:panelHeader id="ph1"
                 text="#{bindings.allEmployeesIterator.hints['categories']['Person'].name}">
    <dynamic:form value="#{bindings.allEmployeesIterator}" id="df1" category="Person"/>
  </af:panelHeader>
  <af:panelHeader id="ph2"
                         text="#{bindings.allEmployeesIterator.hints['categories']['Job'].name}">
    <dynamic:form value="#{bindings.allEmployeesIterator}" id="df2" category="Job"/>
  </af:panelHeader>
  <af:panelHeader id="ph3"
                         text="#{bindings.allEmployeesIterator.hints['categories']['HR'].name}">
     <dynamic:form value="#{bindings.allEmployeesIterator}" id="df3" category="HR"/>
       <af:commandButton text="Submit" id="cb3"/>
  </af:panelHeader>
</af:panelGroupLayout>
```

Just in case you are interested, the hints['categories'] expression returns a HashMap of `CategoryImpl` objects. The `CategoryImpl` exposes a property for **name**. Because the EL expression builder in Oracle JDeveloper doesn't expose the hint property in full detail for the DCIteratorBinding, you need to type this EL into the **text** property yourself.

Note that the UI preview in the visual editor in Oracle JDeveloper will change to {…} for the title strings.

Accessing View Object UI hints from EL

View Objects allow developers to define UI hints in the **General** category. In below screen shot, the **Display Name**, **Display Name (Plural)** and **Description** are defined for the Employees View Object.



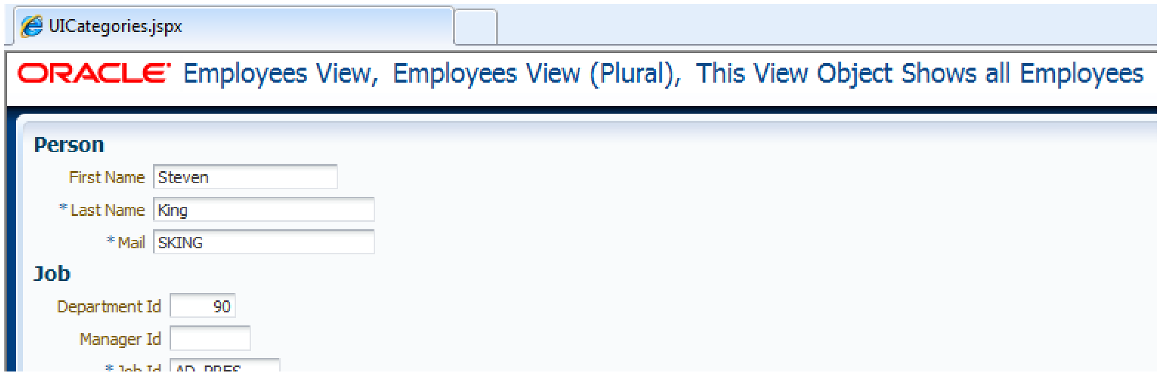
View Object
View objects are for joining, filtering, projecting, and sorting your business data for the specific needs of a given application task.

Name:	EmployeesView
Display Name:	Employees View
Package:	adf.sample.model.vo
Extends:	<None>
Property Set:	<None>
Business Components Project:	Model

UI Hints

Display Name:	Employees View
Display Name (Plural):	Employees View (Plural)
Description:	This View Object Shows all Employees

To display this information on the ADF Faces view as shown below …



ORACLE Employees View, Employees View (Plural), This View Object Shows all Employees

Person

First Name	Steven
* Last Name	King
* Mail	SKING

Job

Department Id	90
Manager Id	
* Job Id	AD_PRES

You use EL expression similar to the one shown the page snippet below

```
<af:panelGroupLayout id="pgl2" layout="horizontal">
   <af:outputText value="#{bindings.allEmployeesIterator.hints['label']}," id="ot3"/>
  <af:spacer width="10" height="10" id="s1"/>
   <af:outputText value="#{bindings.allEmployeesIterator.hints['labelPlural']}, " id="ot2"/>
  <af:spacer width="10" height="10" id="s2"/>
   <af:outputText value="#{bindings.allEmployeesIterator.hints['TOOLTIP']} " id="ot1"/>
</af:panelGroupLayout>
```

So the Display Name property maps to UI hints "label", the Display Name (Plural) property to the UI hints "labelPlural" and the Description property to the "TOOLTIP" string.

These strings are defined in the `oracle.jbo.AttributeHints` class. For future-save programming you should consider reading the UI hint strings directly from this class.

To do so, expose the `AttributeHints` class to EL through a helper managed bean (put it to **application** scope for best performance) that exposes JavaBean properties for each of the strings.

Using such a helper bean, the above page snippets would look similar to this

```
 <af:panelGroupLayout id="pgl2" layout="horizontal">
   <af:outputText value="#{bindings.allEmployeesIterator.hints[helperBean.label]}," id="ot3"/>
  <af:spacer width="10" height="10" id="s1"/>
   <af:outputText value="#{bindings.allEmployeesIterator.hints[helperBean.labelPlural]}, "
                        id="ot2"/>
  <af:spacer width="10" height="10" id="s2"/>
….
```

JSF 2.0 Preemptive Navigation in ADFc of JDeveloper 11.1.2

Preemptive navigation is a new feature in JavaServer Faces 2.0 and allows runtime introspection of control flow cases for their target view.

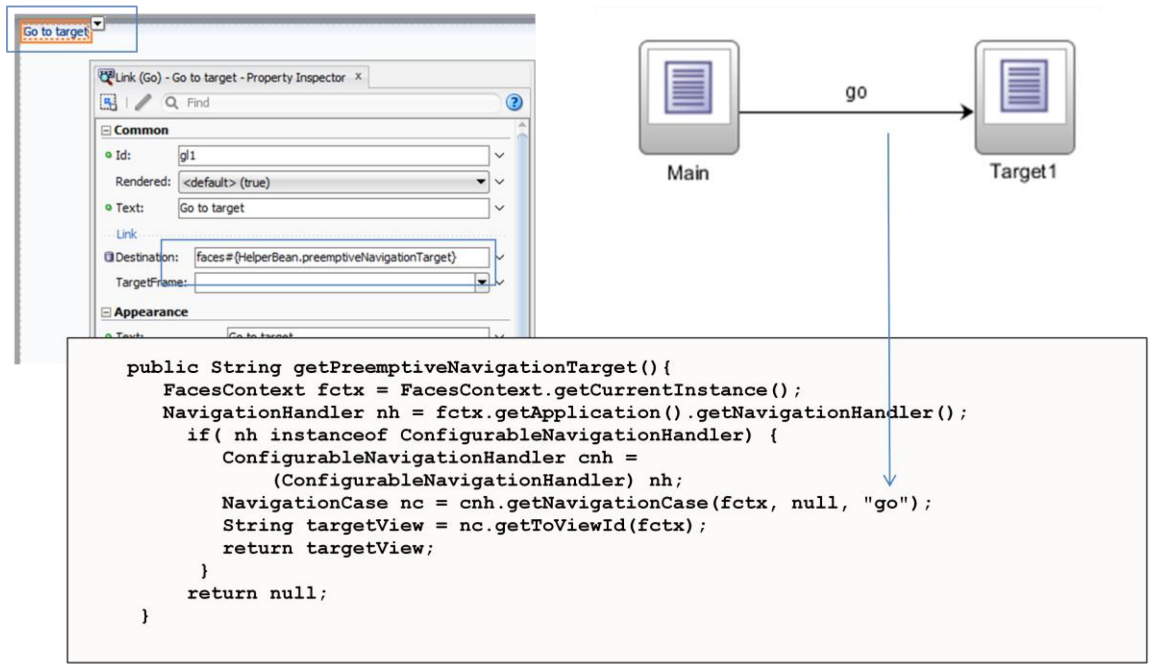
The JSF API for this is the `ConfigurableNavigationHander` class that exposes the following methods

- getNavigationCase(FacesContext context,
                    java.lang.String fromAction,
                    java.lang.String outcome)
- getNavigationCases() – returns a `Map<String, Set<NavigationCase>>` that lists all available navigation cases with the **viewId** as the map keys
- performNavigation(java.lang.String outcome) – Navigates to the next view based on the outcome. Developers using this method must ensure it is used during JSF InvokeApplication phase as it cannot be used any later

The `NavigationCase` class wraps the information defined for a navigation, including the condition (also a new feature in JSF 2.0) in which the navigation case is valid.

Preemptive navigation can be used in an application to populate redirect components, like the goLink shown below with a target view, or for redirects in a managed bean, for which developers need to know the target view.

The sample below shows an ADF Faces **GoLink** pointing to a managed bean. The managed bean returns the redirect URL the link follows when clicked on.



Common
Id: gl1
Rendered: <default> (true)
Text: Go to target
Link
Destination: faces#{HelperBean.preemptiveNavigationTarget}
TargetFrame:
Appearance

```
public String getPreemptiveNavigationTarget(){
   FacesContext fctx = FacesContext.getCurrentInstance();
   NavigationHandler nh = fctx.getApplication().getNavigationHandler();
     if( nh instanceof ConfigurableNavigationHandler) {
        ConfigurableNavigationHandler cnh =
            (ConfigurableNavigationHandler) nh;
        NavigationCase nc = cnh.getNavigationCase(fctx, null, "go");
        String targetView = nc.getToViewId(fctx);
        return targetView;
      }
     return null;
 }
```

The managed bean accesses the `NavigationHandler` defined for the JSF instance and verifies it to be an instance of `ConfigurableNavigationHandler` before it looks up the target viewId for the control flow case.

Note that ADFc in Oracle JDeveloper 11g R2 also supports conditional navigation, in which case developers can define an EL expression on the control flow case, using the Property Inspector, that determines when a navigation case is valid and when it is not.

Also note that Preemptive navigation fails with a **NullPointer** exception if the referenced control flow case is conditionally set to disabled. To handle this, the managed bean code above needs to be surrounded with a **try…catch** block.

How-to read train stop display names from a message bundle

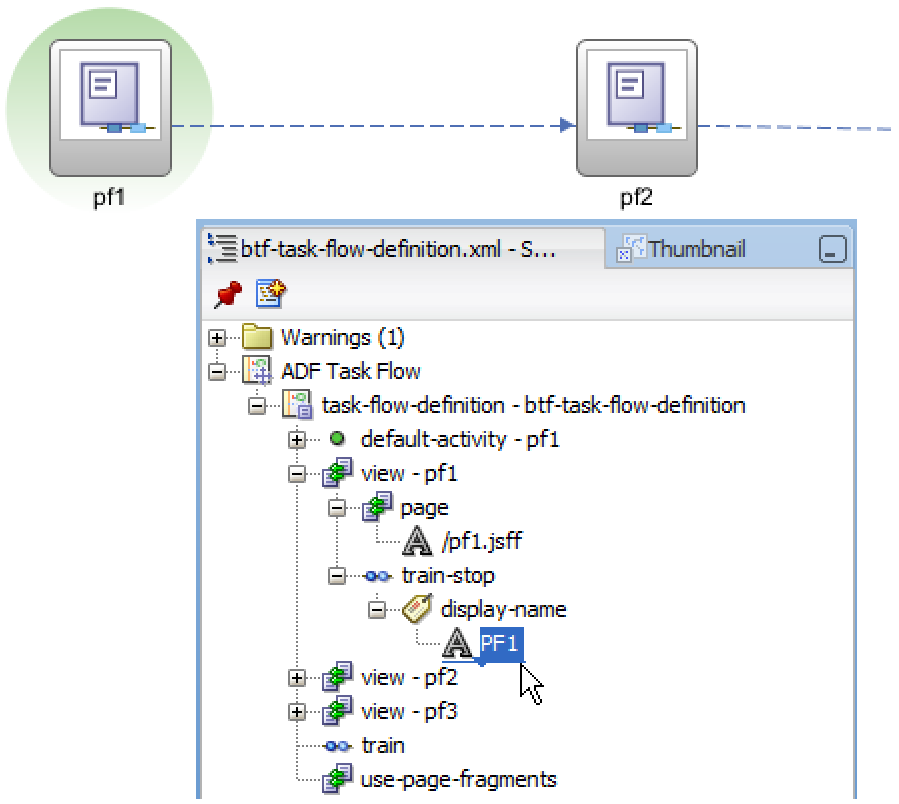
In Oracle JDeveloper 11g R1, you set the display name of a train stop of an ADF bounded task flow train model by using the Oracle JDeveloper Structure Window.

To do so

- Double-click onto the bounded task flow configuration file (XML) located in the Application Navigator so the task flow diagram opens.

- In the task flow diagram, select the view activity node for which you want to define the display name.
- In the Structure Window., expand the view activity node and then the train-stop node therein
- Add the display name element by using the right-click context menu on the train-stop node, selecting **Insert inside train-stop -> Display Name**
- Edit the Display Name value with the Property Inspector

Following the steps outlined above, you can define static display names – like "PF1" for page fragment 1 shown in the image below - for train stops to show at runtime.

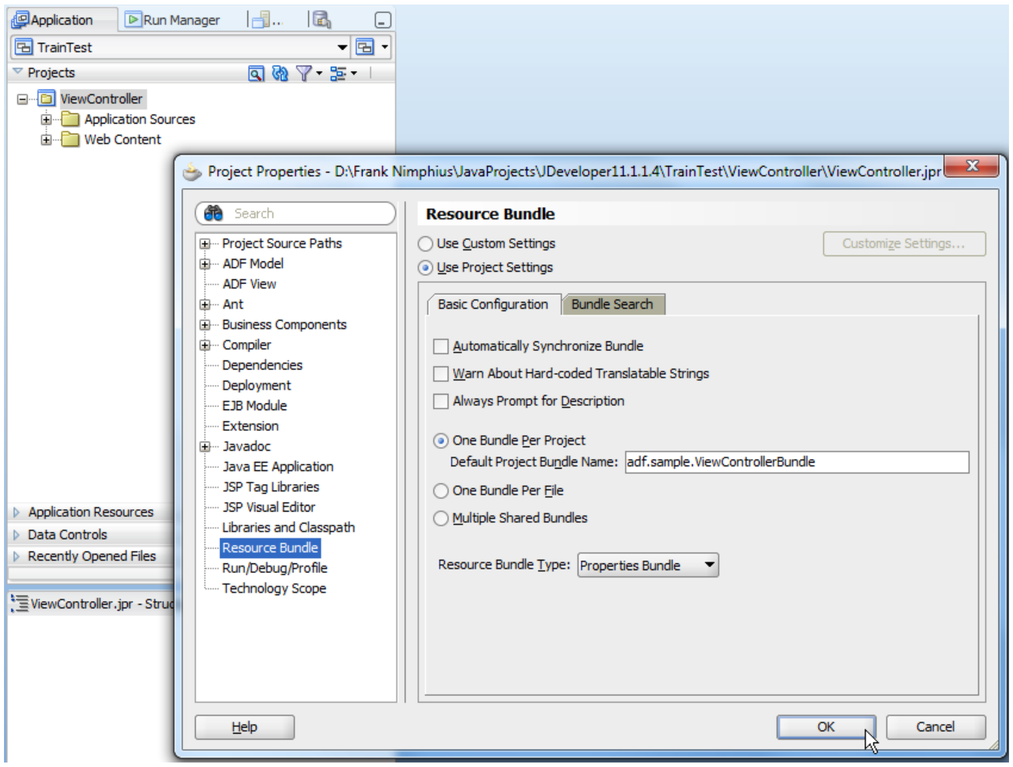


```
Warnings (1)
ADF Task Flow
  task-flow-definition - btf-task-flow-definition
    default-activity - pf1
    view - pf1
      page
        /pf1.jsff
      train-stop
        display-name
          PF1
    view - pf2
    view - pf3
    train
    use-page-fragments
```

In the following, I explain how you can change the static display string to a dynamic string that reads the display label from a resource bundle so train stop labels can be internationalized.

There are different strategies available for managing message bundles within an Oracle JDeveloper project. For this sample, I decided to build and configure the default properties file as indicated by the projects properties. To learn about the project's suggested file name and location, open the JDeveloper project properties (use a right mouse click on the project node in the Application Navigator and choose **Project Properties**.

Select the **Resource Bundle** node to see the suggested name and location for the default message bundle. Note that this is the resource bundle that Oracle JDeveloper would automatically create when you assign a text resource to an ADF Faces component in a page.



Project Properties - D:\Frank Nimphius\JavaProjects\JDeveloper11.1.1.4\TrainTest\ViewController\ViewController.jpr

Resource Bundle

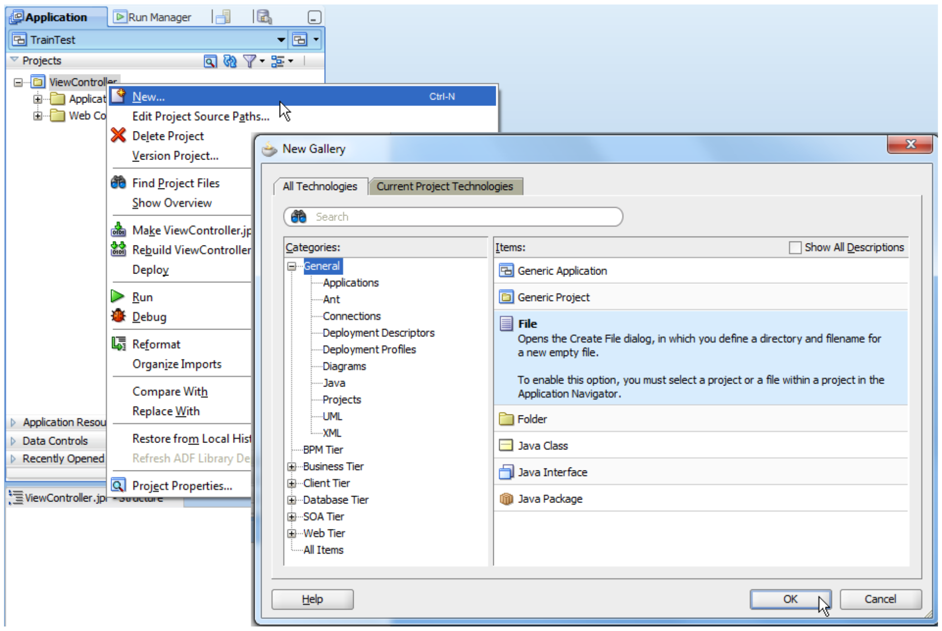
( ) Use Custom Settings
[✓ Use Project Settings]

[ ] Automatically Synchronize Bundle
[ ] Warn About Hard-coded Translatable Strings
[ ] Always Prompt for Description

[✓ One Bundle Per Project]
Default Project Bundle Name: adf.sample.ViewControllerBundle
( ) One Bundle Per File
( ) Multiple Shared Bundles

Resource Bundle Type: Properties Bundle

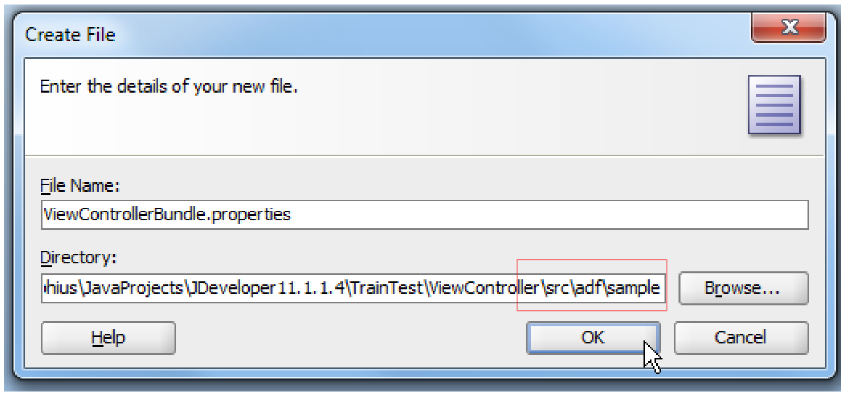
For the train stop display name, we need to create the message bundle manually as there is no context menu help available in Oracle JDeveloper. For this, use a right mouse click on the JDeveloper project and choose **New → General → File** from the menu and in the opened dialog.



File
Opens the Create File dialog, in which you define a directory and filename for a new empty file.

To enable this option, you must select a project or a file within a project in the Application Navigator.

Specify the message bundle file name as the name looked up before in the project properties **Resource Bundle** option. Also, ensure that the file is saved in a directory structure that matches the package structure shown in the **Resource Bundle** dialog. In the sample, the properties file is saved in the View Project's **src → adf → sample** directory.



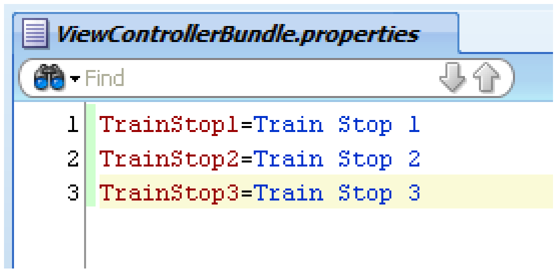
Enter the details of your new file.

File Name: ViewControllerBundle.properties

Directory: hius\JavaProjects\JDeveloper11.1.1.4\TrainTest\ViewController\src\adf\sample

Edit the properties file and define key – values pairs for the train stop component. In the sample, such key value pairs are

TrainStop1=Train Stop 1
TrainStop2=Train Stop 2
TrainStop3=Train Stop 3

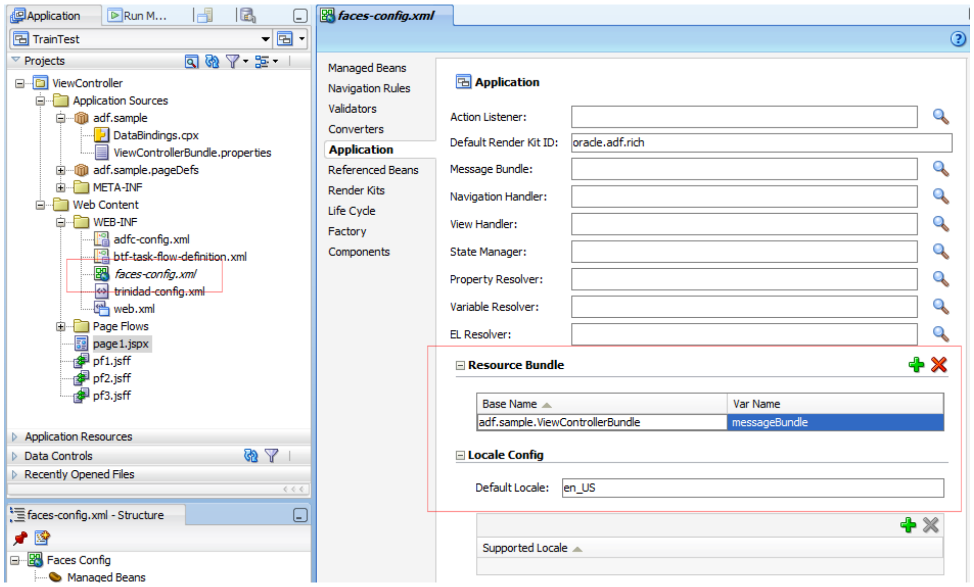


```
TrainStop1=Train Stop 1
TrainStop2=Train Stop 2
TrainStop3=Train Stop 3
```

Next, double click the **faces-config.xml** file and switch the opened editor to the **Overview** tab. Select the **Application** category and press the green plus icon next to the **Resource Bundle** section.

Define the resource bundle **Base Name** as the package and properties file name, for example

adf.sample.ViewControllerBundle



Application

Action Listener:
Default Render Kit ID: oracle.adf.rich
Message Bundle:
Navigation Handler:
View Handler:
State Manager:
Property Resolver:
Variable Resolver:
EL Resolver:

Resource Bundle

Base Name	Var Name
adf.sample.ViewControllerBundle	messageBundle

Locale Config

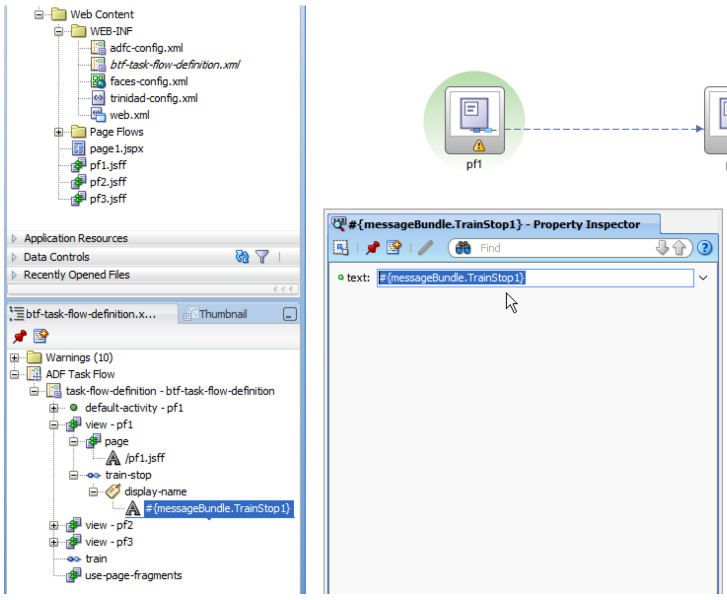
Default Locale: en_US

```
<resource-bundle>
   <base-name>adf.sample.ViewControllerBundle</base-name>
   <var>messageBundle</var>
 </resource-bundle>
```

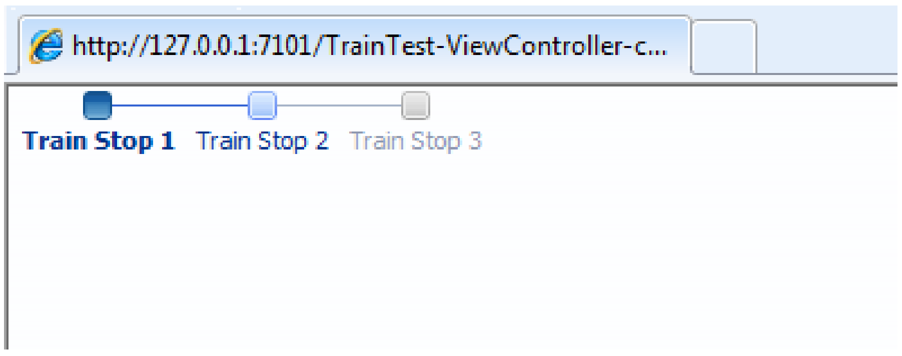
Finally, define a variable name for the message bundle so the bundle can be accessed from Expression Language. For this example, the name is chosen as "messageBundle".

Next, select the **display-name** element in the train stop node (similar to when creating the display name) and use the Property Inspector to change the static string to an EL expression referencing the message bundle, for example:

#{messageBundle.TrainStop1}



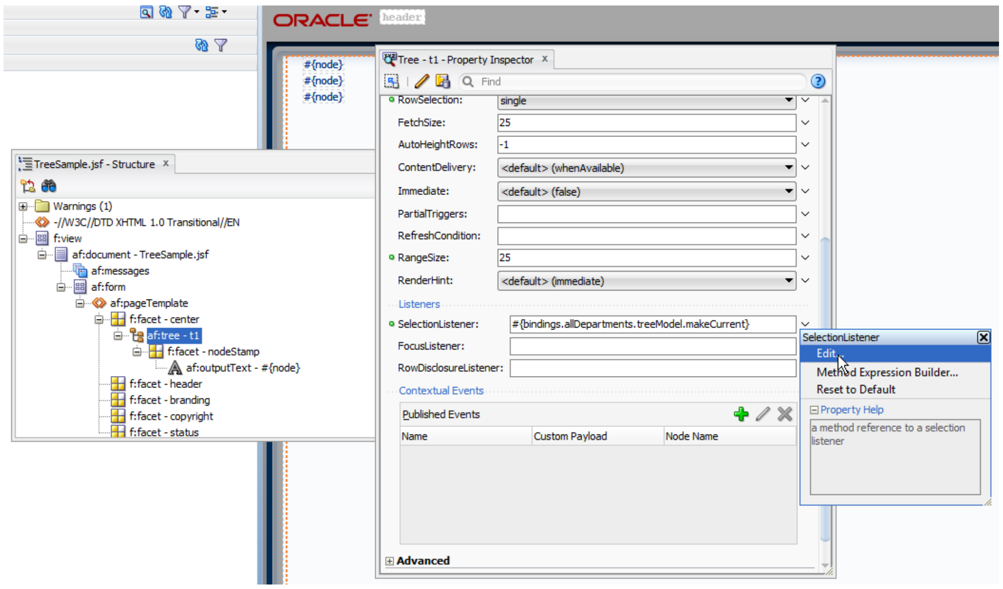
At runtime, the train stops now show display names read from a message bundle.



How-to access selected node in tree component

By default, the **SelectionListener** property of an ADF bound tree points to the **makeCurrent** method of the **FacesCtrlHierBinding** class in ADF to synchronize the current row in the ADF binding layer with the selected tree node. To customize the selection behavior, or just to read the selected node value in Java, you override the default configuration with an EL string pointing to a managed bean method property. In the following I show how you change the selection listener while preserving the default ADF selection behavior.

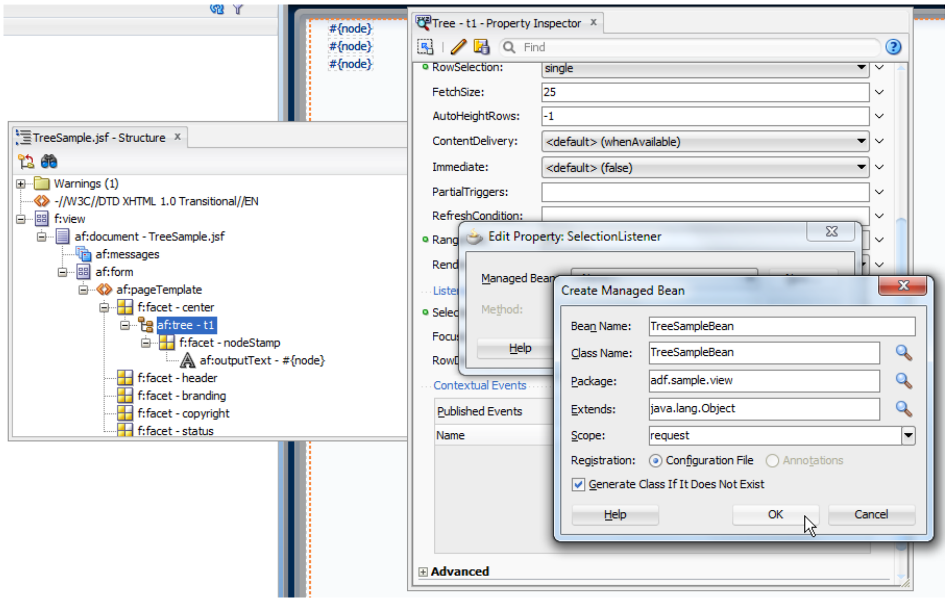
To change the **SelectionListener**, select the tree component in the Structure Window and open the Oracle JDeveloper Property Inspector. From the context menu, select the **Edit** option to create a new listener method in a new or an existing managed bean.



Tree - t1 - Property Inspector

RowSelection:	single
FetchSize:	25
AutoHeightRows:	-1
ContentDelivery:	<default> (whenAvailable)
Immediate:	<default> (false)
PartialTriggers:	
RefreshCondition:	
RangeSize:	25
RenderHint:	<default> (immediate)
Listeners	
SelectionListener:	#{bindings.allDepartments.treeModel.makeCurrent}
FocusListener:	
RowDisclosureListener:	

For this example, I created a new managed bean



Create Managed Bean

Bean Name:	TreeSampleBean
Class Name:	TreeSampleBean
Package:	adf.sample.view
Extends:	java.lang.Object
Scope:	request
Registration:	[✓ Configuration File] Annotations
	✓ Generate Class If It Does Not Exist

On tree node select, the managed bean code prints the selected tree node value(s)

```
import java.util.Iterator;
import java.util.List;

import javax.el.ELContext;
import javax.el.ExpressionFactory;
import javax.el.MethodExpression;
import javax.faces.application.Application;
import javax.faces.context.FacesContext;

import oracle.adf.view.rich.component.rich.data.RichTree;
import oracle.jbo.Row;
import oracle.jbo.uicli.binding.JUCtrlHierBinding;
import oracle.jbo.uicli.binding.JUCtrlHierNodeBinding;

import org.apache.myfaces.trinidad.event.SelectionEvent;
import org.apache.myfaces.trinidad.model.CollectionModel;
import org.apache.myfaces.trinidad.model.RowKeySet;
import org.apache.myfaces.trinidad.model.TreeModel;


public class TreeSampleBean {

    public TreeSampleBean() {

    }


 public void onTreeSelect(SelectionEvent selectionEvent) {
   //original selection listener set by ADF
   //#{bindings.allDepartments.treeModel.makeCurrent}
   String adfSelectionListener =
           "#{bindings.allDepartments.treeModel.makeCurrent}";

   //make sure the default selection listener functionality is
   //preserved. you don't need to do this for multi select trees
   //as the ADF binding only supports single current row selection

   /* START PRESERVER DEFAULT ADF SELECT BEHAVIOR */
   FacesContext fctx = FacesContext.getCurrentInstance();
   Application application = fctx.getApplication();
   ELContext elCtx = fctx.getELContext();
   ExpressionFactory exprFactory = application.getExpressionFactory();

   MethodExpression me = null;
   me =  exprFactory.createMethodExpression(
                       elCtx, adfSelectionListener,
                       Object.class, newClass[]{SelectionEvent.class});
   me.invoke(elCtx, new Object[] { selectionEvent });

   /* END PRESERVER DEFAULT ADF SELECT BEHAVIOR */
   RichTree tree = (RichTree)selectionEvent.getSource();
   TreeModel model = (TreeModel)tree.getValue();

   //get selected nodes
   RowKeySet rowKeySet = selectionEvent.getAddedSet();
   Iterator rksIterator = rowKeySet.iterator();
   //for single select configurations, thi sonly is called once
   while (rksIterator.hasNext()) {
    List key = (List)rksIterator.next();
    JUCtrlHierBinding treeBinding = null;
    CollectionModel collectionModel = (CollectionModel)tree.getValue();
    treeBinding = (JUCtrlHierBinding)collectionModel.getWrappedData();
    JUCtrlHierNodeBinding nodeBinding = null;
    nodeBinding = treeBinding.findNodeByKeyPath(key);
    Row rw = nodeBinding.getRow();
    //print first row attribute. Note that in a tree you have to
    // determine the node type if you want to select node attributes
    //by name and not index

    String rowType = rw.getStructureDef().getDefName();

    if(rowType.equalsIgnoreCase("DepartmentsView")){
      System.out.println("This row is a department: " +
                          rw.getAttribute("DepartmentId"));

     }
     else if(rowType.equalsIgnoreCase("EmployeesView")){
       System.out.println("This row is an employee: " +
                           rw.getAttribute("EmployeeId"));

      }
     else{
       System.out.println("Huh ????");
     }
     // ... do more useful stuff here
   }
 }
}
```

How-to create a command button at runtime

In ADF Faces, the command button is an instance of `RichCommandButton` and can be created at runtime. While creating the button is not difficult at all, adding behavior to it requires knowing about how to dynamically create and add an action listener reference. The example code below shows two methods: The first method, **handleButtonPress** is a public method exposed on a managed bean.

```
public void handleButtonPress(ActionEvent event){
   System.out.println("Event handled");
   //optional: partially refresh changed components if command
   //issued as a partial submit
}
```

The second method is called in response to a user interaction or on page load and dynamically creates and adds a command button. When the button is pressed, the managed bean method – the action handler – defined above is called. The action handler is referenced using EL in the created `MethodExpression` instance. If the managed bean is in viewScope, backingBeanScope or pageFlowsScope, then you need to add these scopes as a prefix to the EL (as you would when configuring the managed bean reference at design time)

```
//Create command button and add it as a child to the parent component that is passed as an
//argument to this method
 private void createCommandButton(UIComponent parent){
     RichCommandButton edit = new RichCommandButton();
     //make the request partial
     edit.setPartialSubmit(true);
     edit.setText("Edit");

      //compose method expression to invoke action event handler when button is
      //pressed
     FacesContext fctx = FacesContext.getCurrentInstance();
      Application application = fctx.getApplication();
      ExpressionFactory elFactory = application.getExpressionFactory();
      ELContext elContext = facesCtx.getELContext();

      MethodExpression methodExpression =null;
      //Make sure the EL expression references a valid managed bean method. Ensure
      //the bean scope is properly addressed
       methodExpression = elFactory.createMethodExpression(
                                  elContext, "#{myRequestScopeBean.handleButtonPress}",
                                  Object.class, new Class[] {ActionEvent.class});

      //Create the command button action listener reference
       MethodExpressionActionListener al = null;

       al= new MethodExpressionActionListener(methodExpression);
       edit.addActionListener(al);


       //add new command button to parent component and PPR the component for
       //the button to show
        parent.getChildren().add(edit);
       AdfFacesContext adfFacesContext = AdfFacesContext.getCurrentInstance();
       adfFacesContext.addPartialTarget(parent);
    }
```

Accessing the JSESSIONID from JSF

The following code attempts to access and print the user session ID from ADF Faces, using the session cookie that is automatically set by the server and the Http Session object itself.

```
FacesContext fctx = FacesContext.getCurrentInstance();
ExternalContext ectx = fctx.getExternalContext();
HttpSession session = (HttpSession) ectx.getSession(false);

String sessionId = session.getId();
System.out.println("Session Id = "+ sessionId);

Cookie[] cookies =
          ((HttpServletRequest)ectx.getRequest()).getCookies();

//reset session string
sessionId = null;

if (cookies != null) {
  for (Cookie brezel : cookies) {
     if (brezel.getName().equalsIgnoreCase("JSESSIONID")) {
       sessionId = brezel.getValue();
        break;
      }
    }
 }
 System.out.println("JSESSIONID cookie = "+sessionId);
```

Though apparently both approaches to the same thing, they are different in the value they return and the condition under which they work. The getId method, for example returns a session value as shown below

grLFTNzJhhnQTqVwxHMGl0WDZPGhZFl2m0JS5SyYVmZqvrfghFxy!-1834097692!1322120041091

Reading the cookie, returns a value like this

grLFTNzJhhnQTqVwxHMGl0WDZPGhZFl2m0JS5SyYVmZqvrfghFxy!-1834097692

Though both seem to be identical, the difference is within "*!1322120041091*" added to the id when reading it directly from the Http Session object. Dependent on the use case the session Id is looked up for, the difference may not be important.

Another difference however, is of importance. The cookie reading only works if the session Id is added as a cookie to the request, which is configurable for applications in the `weblogic-application.xml` file. If cookies are disabled, then the server adds the session ID to the request URL (actually it appends it to the end of the URI, so right after the view Id reference). In this case however no cookie is set so that the lookup returns empty. In both cases however, the **getId** variant works.

How to detect browser type and version from ADF Faces

Sometimes ADF applications need to know about the user browser type and version. For this, assuming you need this information in Java, you can use the Trinidad `RequestContext` object. You could also use the `AdfFacesContext` object for the same, but since the ADF Faces `Agent` class is marked as **deprecated**, using the equivalent Trinidad classes is the better choice.

The source code below prints the user browser information to the Oracle JDeveloper message window

```
import org.apache.myfaces.trinidad.context.Agent;
import org.apache.myfaces.trinidad.context.RequestContext;

…

RequestContext requestCtx = RequestContext.getCurrentInstance();
Agent agent = requestCtx.getAgent();
String version = agent.getAgentVersion();
String browser = agent.getAgentName();
String platform = agent.getPlatformName();
String platformVersion = agent.getPlatformVersion();

System.out.println("==================");
System.out.println("Your browser information: ");
System.out.println("Browser: "+browser);
System.out.println("Browser Version : "+version);
System.out.println("Browser Platform: "+platform);
System.out.println("Browser Platform Version: 
                   "+platformVersion);
System.out.println("==================");
```

Ps.: I can almost hear John Stegeman, the Oracle ADF community expert I introduce in this month's OTN Harvest Spotlight, saying that `System.out.println` is not what I should be using, but `AdfLogger`. And right he is – however, samples fall under a different category than production systems.

ADF Code Corner

OTN Harvest Spotlight

- John Stegeman

ORACLE CODE CORNER ADF OTN HARVEST

- paragraph about person –

I've been working in IT for the past 21 years. I was born in the USA, but I
I am currently living and working in the UK. I'm married to my beautiful wife Josephine and have three great children, Purity, David, and Matthew.

"select answer from forum_answers where context_provided is null and question_is_not_related_to_thread = 'TRUE' "

- John Stegeman

ACC:	What is your current role?
John:	I am an architect / presales resource in the insurance software group of Xchanging, a BPO/ITO company headquartered in London.
ACC:	What is your IT background?
John:	I don't have any formal IT education (I was a chemistry/chemical engineering major in university), but starting from my very first work-study experience, I was doing programming tasks in Fortran to simulate chemical production efficiencies. I started my career in IT in 1990 and have been working with Oracle technology in one way or another since then. I started using JDeveloper in about 1999, shortly after Oracle bought the source code for JBuilder from Borland.
ACC:	How do you currently use Oracle JDeveloper and ADF?
John:	Actually, I'm not currently using Oracle JDeveloper or ADF in any projects. When I do use them, I've used them to build internal applications for our customers. Although I'm not working on any ADF projects at the moment, I still invest a lot of time keeping current so that I don't get rusty and am ready for the next opportunity that comes up.
ACC:	So far, what has been your biggest challenge in building Java EE application with Oracle ADF?
John:	The biggest challenge that I've had is taking teams of people that understand and have lived

and breathed in Oracle Forms and helping them be productive in Oracle ADF.

At the time I built my first real live ADF application, there was very little experience in the marketplace (my first app was in JDeveloper 10.1.3 – the first version to have ADF Faces), so learning the "best practices" was a matter of building them as we went along.

ACC: Which feature of ADF was the greatest benefit to your project?

John: The feature that was the best from my perspective was the ADF binding layer. Not having to worry about how to link up the data to the UI was a big productivity booster, and made things that much easier for the Forms developers to pick up.

ACC: Away from the on line help, what have been your most valuable sources of ADF knowledge?

John: I learned tons from the OTN Forums, both from asking questions in my early days and more in answering questions of late.

I am by nature a person who learns by solving problems, so answering questions on the Forum was and is very beneficial. The Oracle documentation (the Fusion Developer's Guide for ADF) is also something that was very helpful to me.

Even though I've read it cover-to-cover, I still find gems in there that I'd read before but forgotten because I didn't need to know them at the moment.

ACC: Are you in any way actively involved in the ADF Community?

John: I've been a bit less active of late, but I am or have been active on the Forms, in blogging (although it's been a long time since I've blogged anything), presenting to user groups, participating in the ADF Enterprise Methodology Group and writing articles for OTN.

ACC: What is your recommendation for developers new to Oracle ADF. How should they start?

John: This is a tough one, because it depends on the background of the developers.

People who know Forms and people who know Java/J2EE will have a very different learning path to take. I can give a few general suggestions, though:

a) Read the documentation and one or more of the excellent books that are out there.

b) Don't be afraid to experiment – you will learn a lot this way.

c) Find a mentor (or hire a consultant) to help you with your first project. It will really help you focus (ADF is really huge) and avoid learning to do things the wrong way.

d) Spend some time reading on the OTN JDeveloper/ADF forum. I learn a lot from hearing what other people are struggling with and definitely have learned some new things from seeing people's answers.

e) Join the ADF Enterprise Methodology Group.

ACC:	You are very active in answering questions on the Oracle JDeveloper forum. Is there anything people can do to make your life on the forum easier?
John:	That sounds like a trick question – answering questions on the forum isn't part of my job description, so it's not about making ***my*** life easier. I answer questions on the forums for two main reasons: first, I like helping people and solving problems; second, it helps me learn new things. Maybe the question you should have asked is what people can do to make their own lives easier (in other words, how to get help faster)? I have a few suggestions for that one: a). Search before asking – you can often get an answer really fast this way. b). Slow down when you ask a question – make sure you give all of the background information and explain yourself clearly. One of my personal pet peeves on the forum is when people make me play Twenty Questions just so I can understand their question. There are some people who do a really good job of asking questions, whereas others post a one-liner with the obligatory PLEASE HELP ME IT'S URGENT!!!!!! at the end ☺
ACC:	Are you attending the UKOUG conference in Brimingham 2011? If so, what are your planned activities there and how could people meet you just to say 'hello'?
John:	Unfortunately, I'm not likely to make it up to UKOUG this year due to some really pressing project needs, but I'm always happy to meet people in the London area for a cup of coffee. The easy way for people to contact me might be to send a direct Twitter message to @stegemanoracle.
ACC:	ADF Genie grants you a wish, what would you ask for?
John:	Hmm, hard to pick just one, but I'd like to see Oracle JDeveloper work really nicely with Maven for ADF projects.
ACC:	Thanks John.

RELATED DOCOMENTATION

⊠	Oracle Magazine Article on ADF Train Model http://www.oracle.com/technetwork/issue-archive/2011/11-sep/o51adf-452576.html
⊠	
⊠	