Ode to the taskflow

Hands-on Part I

Authors
Luc Bors & Lucas Jellema

Creation date
3 mei 2010

Modification date
7 May 2010

Version/ status
1.1

Filename
hands on part I.docx

Document management

Revision history

date	author	version	remarks
5 May 2010	Luc Bors	0.9	
6 May 2010	Luc Bors	1.0	Minor changes after revision by Lucas
7 May 2010	Luc Bors	1.1	Minor changes after session

Contents

No table of figures entries found.

1. My First Taskflow

In this hands on you will create a taskflow that we will use to learn the basics of ADF taskflows. The taskflow will receive your job title, and optionally your favorite color. Based on that information a welcome message is created. You are also able to enter/change your name.

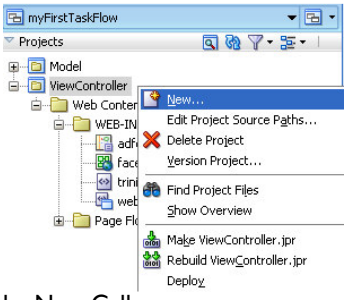
This taskflow will include routers, method calls, input parameters, default activities and wild card control flows. The taskflow that we are going to create will be smart enough to invoke the proper logic.
The taskflow will use a router to invoke the correct page fragment based on an input parameter.

1.1 Open the application

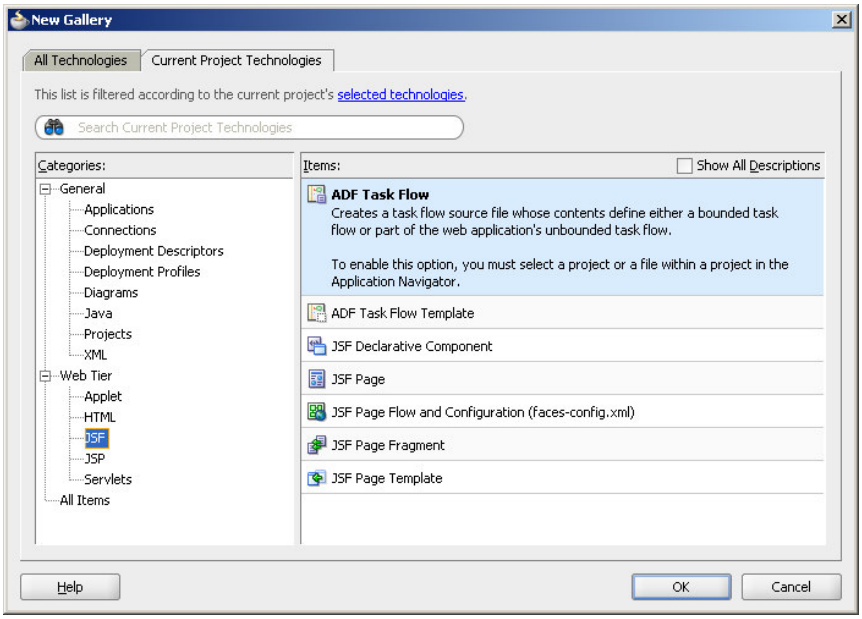
a) You will be provided with a JDeveloper workspace (myFirstTaskFlow.jws) containing only one ADF page. This page will be used to test the taskflow that you will create in this hands-on. Open the Workspace in JDeveloper.

1.2 Create a taskflow with page fragments

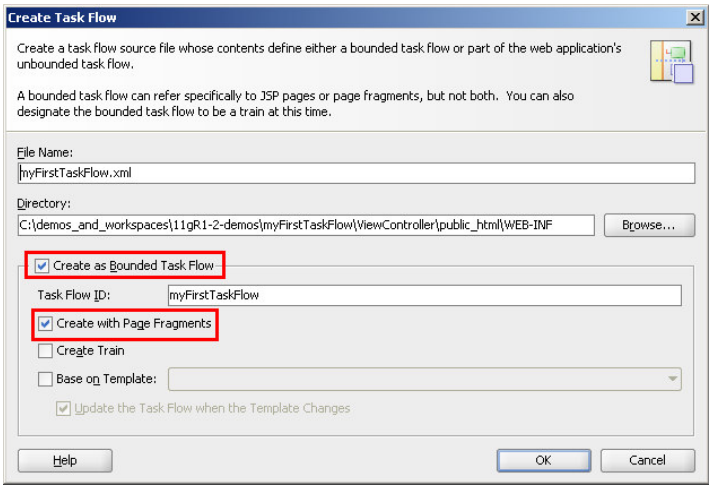
a) Taskflows are created in the context of the ViewController project. So in the ViewController project invoke the right mouse button (rmb) menu and select new……



b) Select ADF Task Flow from the New Gallery.

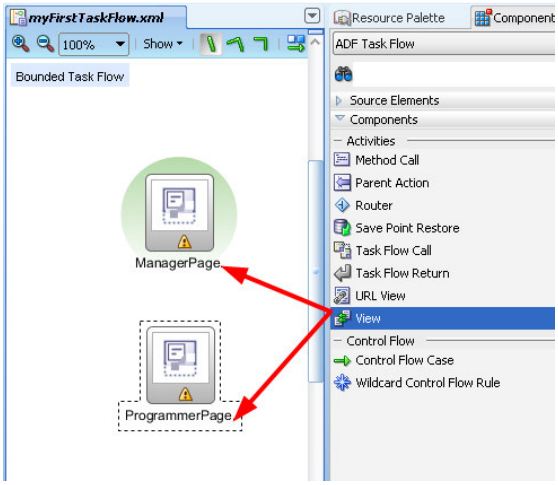


c) The Create Taskflow Wizard is invoked. In this wizard name the taskflow myFirstTaskFlow. Make sure that both "Create as Bounded Task Flow", and "Create with Page Fragments" are checked.



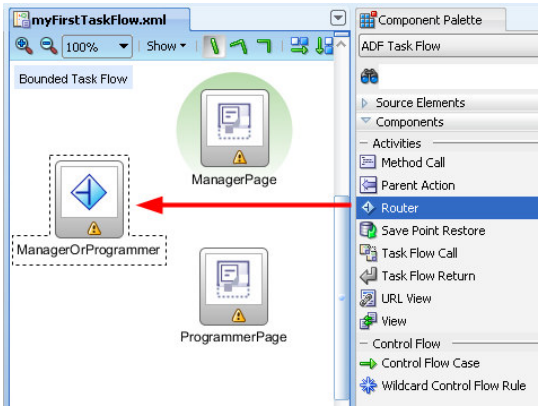
This makes sure that the taskflow is created as a "Bounded Taskflow", and that every view activity in the taskflow will be created as a page fragment instead of a standalone page.

d) The next step in this hands-on is the creation of view activities in the taskflow. These view activities will represent the User interface that will be used in the application consuming the taskflow. From the component pallet, drop a View activity to the taskflow an call it "ManagerPage" . Drop another View activity and call it "ProgrammerPage".

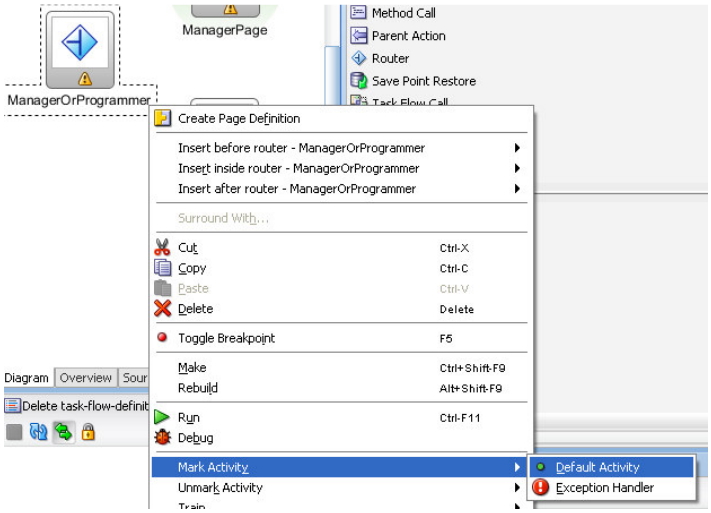


Note the green circle around the "ManagerPage". This indicates that the ManagerPage is the default activity in this taskflow. We will change that later, but for now it is ok. Also note the exclamation marks at both pages. This indicates that there is not a related page or pagefragment available yet. We will create these later on.

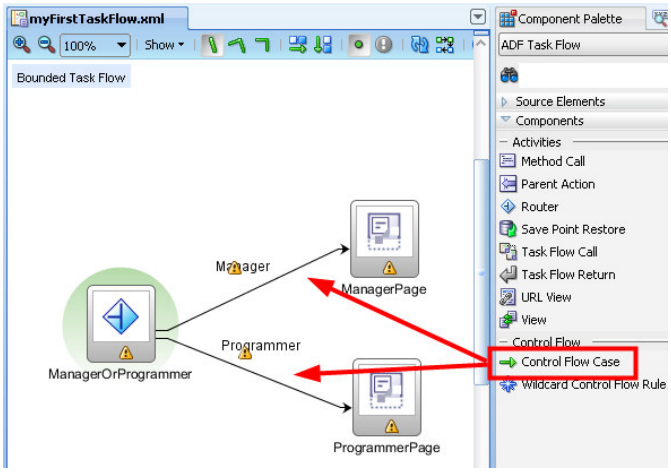
e) Simply adding two view activities to the taskflow is kind of awkward if you have no logic to decide what page to show. The handle this functionality we will add a router to the taskflow. This router will decide where to go: Either to the ManagerPage, or to the ProgrammerPage. Drag a router from the Component Palette to the taskflow. Call the router ManagerOrProgrammer.



f) The Router should be the first activity of this Taskflow, because the router will decide which way we go. Select the router and from the right mouse menu choose "Mark Activity….. Default Activity. Notice that the Green Circle is now on the router. If it's not, something went wrong.

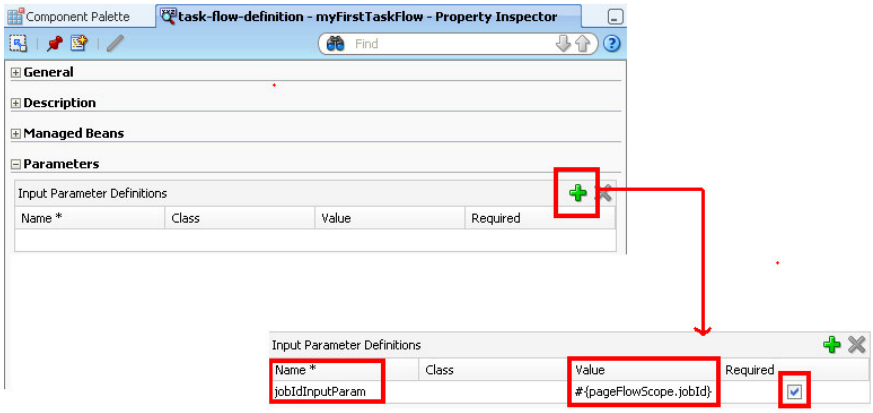


g) With the router as starting point, we now need to add Control Flow Cases from the router to both view activities. From the component palette select "Control Flow Case" and drag it from the Router to the ManagerPage. Drag another one to the ProgrammerPage.



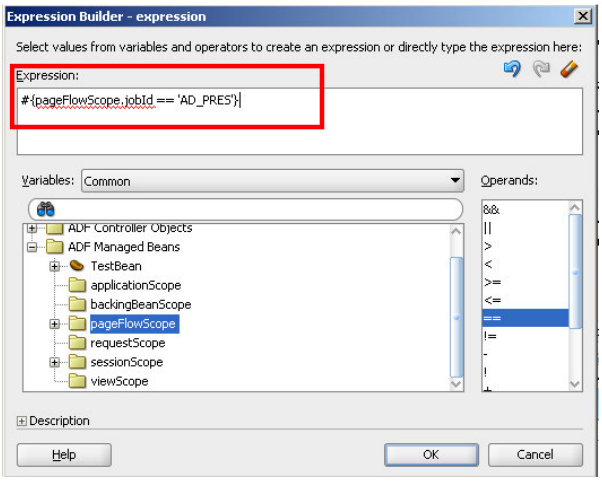
Both Flow Cases have an exclamation mark on them. This is there to indicate that the router does not yet have an outcome with the value "Manager" or "Programmer". This is the next thing we are going to take care of.

h) The router needs a value, so it can decide which way to go. We are going to use an Input Parameter to receive the value when the taskflow is called. Create a new Input Parameter for the taskflow. This can be done in the Parameters section in the Property inspector of the Taskflow by clicking the "add" icon. Make sure to use the correct Name ("jobIdInputParam") and Value ("#{pageFlowScope.jobId}") for the parameter, and check the "Required" checkbox, as this is a required parameter.

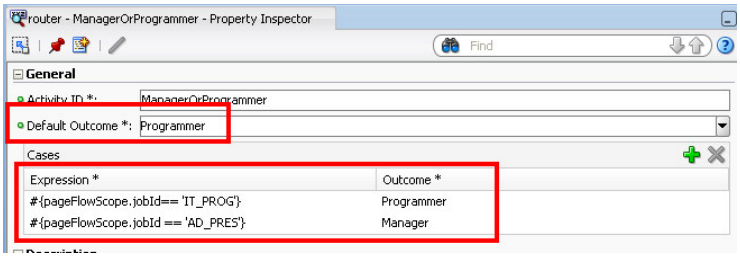


Leaving the Class property empty results in the default Class of java.util.String

i) Now we can adjust the router. Select the router in the Taskflow and go to the property palette. Under the "general" node you can define navigation cases. Invoke the expression builder to create an EL expression to evaluate the value of the Input Parameter.

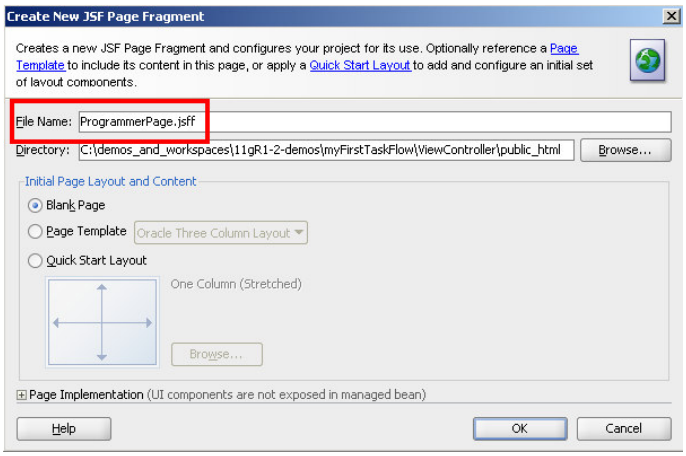


For the IT_PROG, set the Outcome to Programmer. Repeat this step for the manager (evaluating the value to 'AD_PRES'). Now you have defined two outcomes in the router. Finally define the Default Outcome for the taskflow. For now this should be "Programmer".



So far you "only" created a functioning router and two flows on the taskflow. You will certainly need to create page fragments for your taskflow to function. We will do this now.

j) Create the page fragments for the ManagerPage and ProgrammerPage is pretty straightforward. You only need to double-click the view activity on the taskflow. Let's start with the ProgrammerPage. Managers go in second place (as always…..). Double click it to invoke the "create new JSF Page Fragment" wizard. Note the extension of the page fragment, which is "jsff". In the wizard you simply accept all the defaults and click OK.



JDeveloper now creates a new page fragment for you.

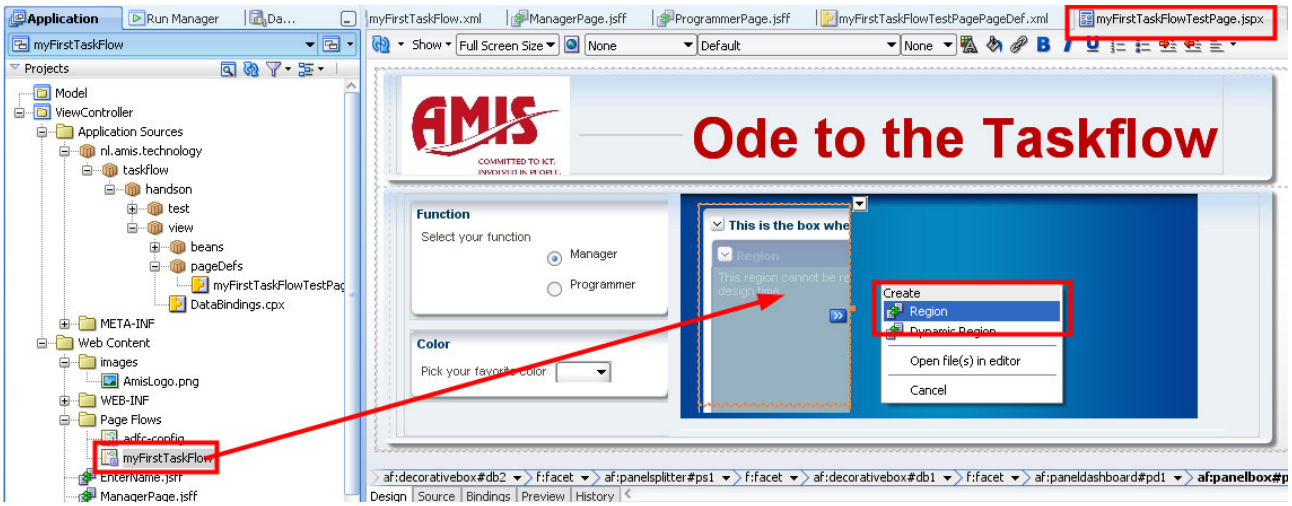
k) To make sure that we can identify this fragment as being the ProgrammerPage, we add an outputText component to it saying that we are on the ProgrammerPage.

l) Repeat the previous two steps for the ManagerPage. Don't forget to set the correct value for the outputText.

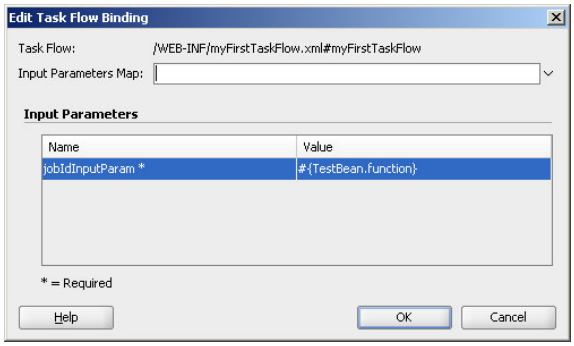
1.3 Preparing the test page

Now we have everything in place to test our taskflow. Do we? Our application needs a page to invoke the taskflow, as the taskflow itself is not run able. Taskflows need to be embedded in JSF pages in order to be executed. Therefore, it is time to add the taskflow to our test page.

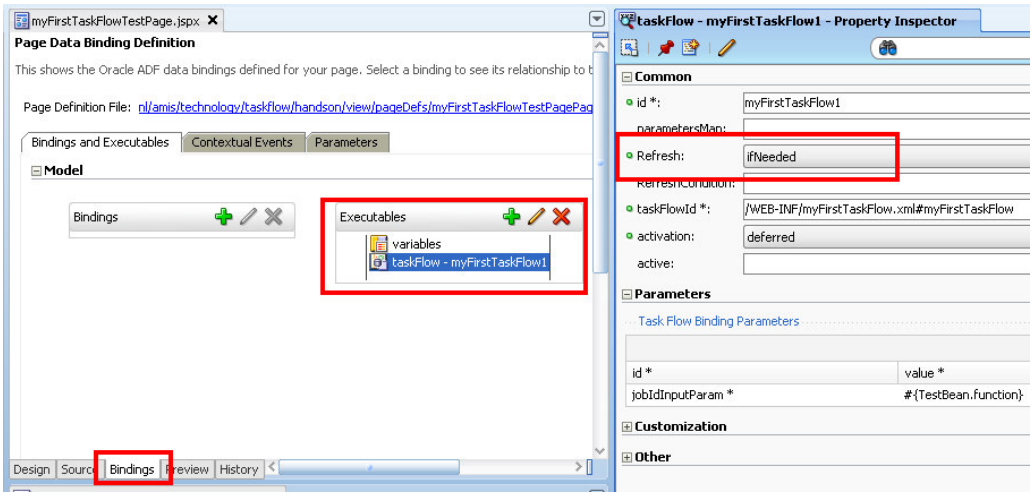
a) Open the test page and drop the taskflow on the page. This will invoke a popup menu in which you can choose between a Region and a Dynamic Region. For our test page we will use a Region.



b) After dropping, a popup appears. JDeveloper knows that the taskflow you just dropped needs an InputParameter, and even knows the name of this parameter. We want the parameter to use the value that we selected in the radiobutton. This value can be found in the testbean. Set the value to #{TestBean.function}.



c) Now we are ready to run the page. Let's do so. Notice that the Managers page fragment is shown. Change the value radio button and you see the Programmers page fragment. Aha, the region is not refreshed! We have to tell the region that it should refresh when the value of its input parameter changes. When you dropped the taskflow on the page, JDeveloper not only created a region, but also created a binding for the region. The taskflow binding needs to be refreshed if the value of the input parameter changes. This can be achieved by setting its 'Refresh' property to 'ifNeeded'.



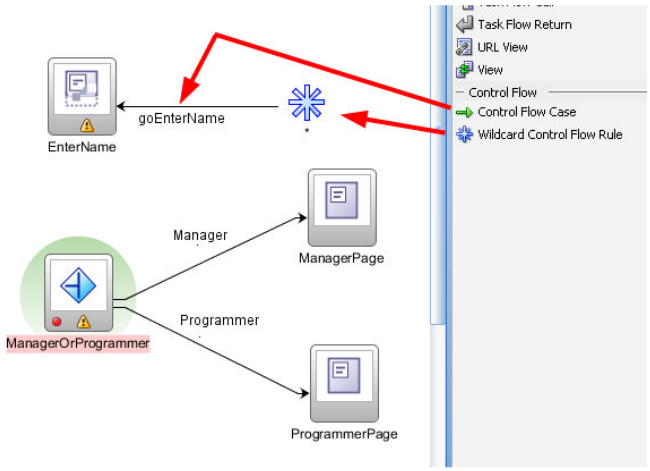
d) Now run the page again and see what happens if you change the value of the radio button.

1.4 Enhancing the taskflow.

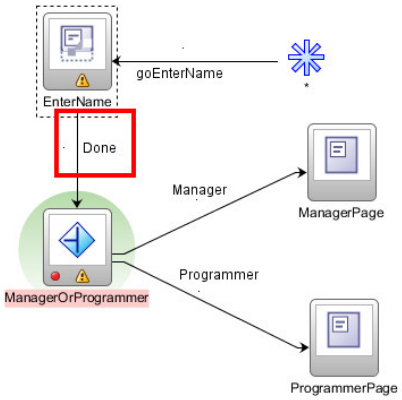
By now you should have a working taskflow with absolutely no functionality. We are going to make some enhancements. Let's assume that the task flow needs some information on the person that is currently using the task flow. We are going to create functionality to enter this information. For this we need a couple of things.

- A new view activity to enter the information
- Control flow handle the flow
- A bean to store the information
- Buttons on all page fragments to invoke navigation

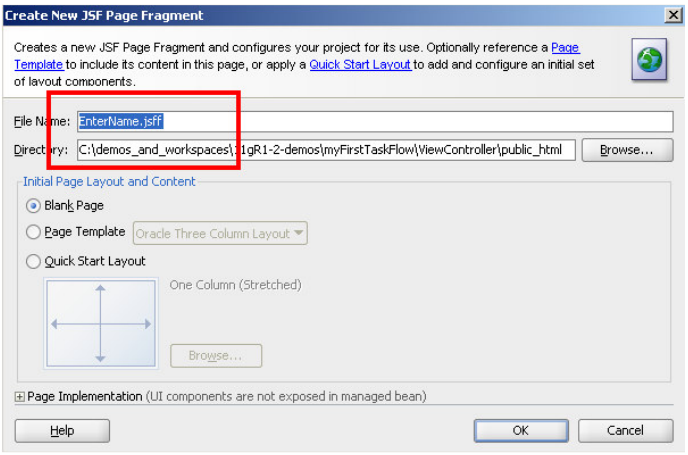
a) Drop a new view activity on the taskflow, and call this activity "EnterName". This activity can be invoked from both, the ManagerPage and the DeveloperPage. To make this happen we will use a 'Wildcard Control Flow Rule'. Drag a "Control Flow Case" form the wildcard to the "EnterName" view activity. Call this Control Flow Case "goEnterName".



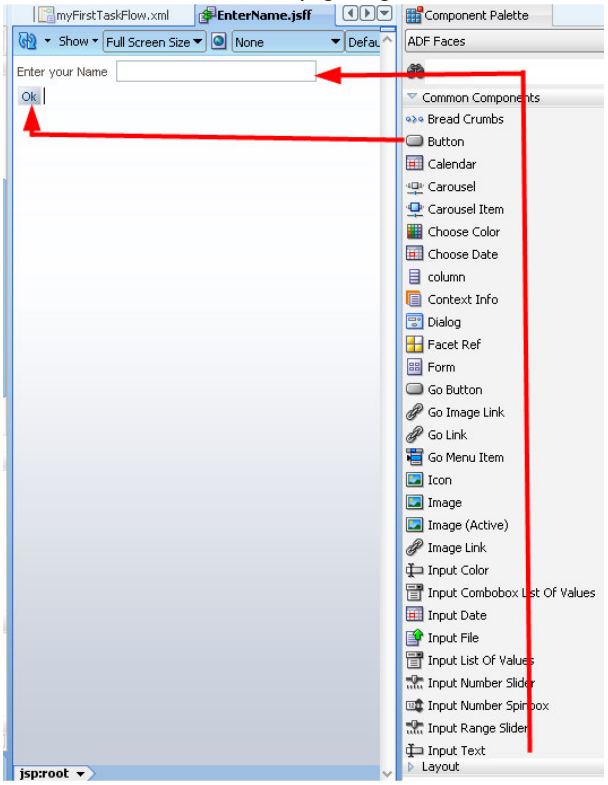
b) We have to make sure that after entering the name the correct page fragment is rendered again. We can re-use the router for that. Create a control flow case from the "EnterName" activity to the router. Call this control flow case "Done".



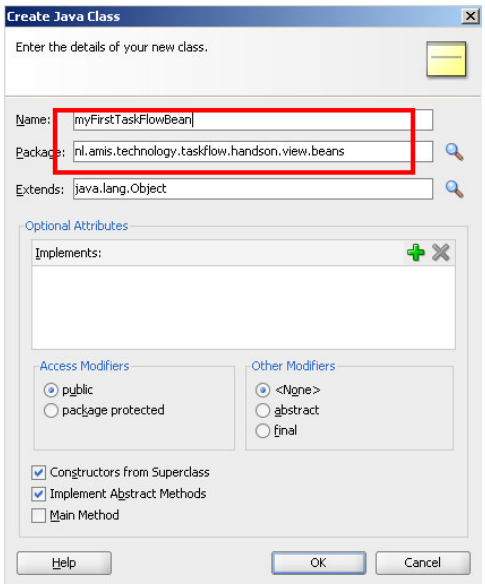
c) Create the page fragment by double clicking on the view activity.



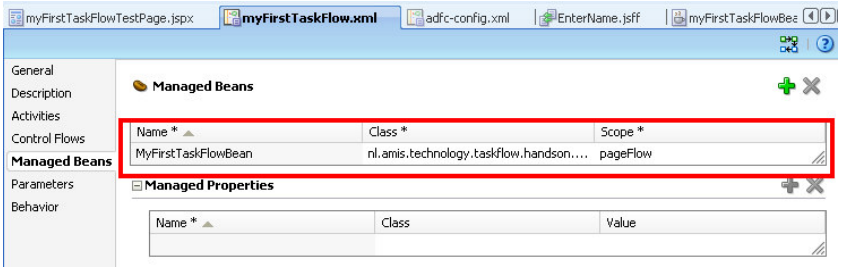
d) Drop an inputText and a commandButton on the page fragment.



e) Now we are going to create a managed bean to hold the value of the inputText. Create a new java class. Enter the class name, and put the class in the package "nl.amis.technology.taskflow.handson.view.beans".



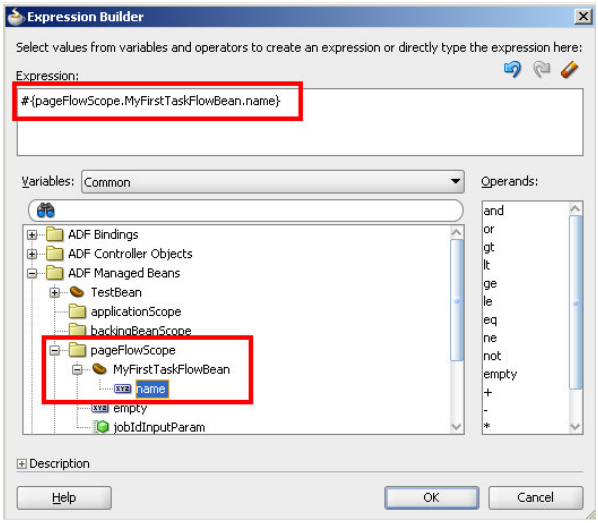
f) Create a new managed bean in the taskflow. Call the bean MyFirstTaskFlowBean and set the Class Name to the class we just created: "nl.amis.technology.taskflow.handson.view. myFirstTaskFlowBean".



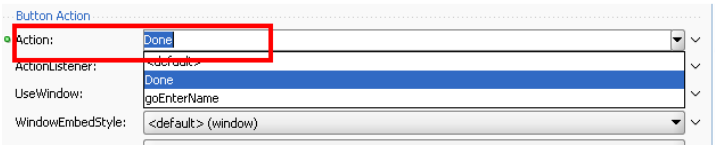
g) Open the java class. Create a bean property "name", and generate assessors for the property.

```
package nl.amis.technology.taskflow.handson.view.beans;

public class myFirstTaskFlowBean {

private String name;
  public myFirstTaskFlowBean() {
  }

  public void setName(String name) {
    this.name = name;
  }

  public String getName() {
    return name;
  }
}
```

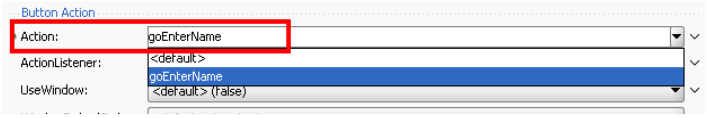
h) Finally go back to the inputTextField and set its value property to "#{pageFlowScope.MyFirstTaskFlowBean.name}".



i) If the Ok button is pressed on the "EnterName" page fragment, the Control Flow "Done" should be executed. To achieve that set the action property of the OK button to "Done".



j) Finally, on both the ManagerPage and the ProgrammerPage create a commandButton to invoke the "goEnterName" Control Flow.



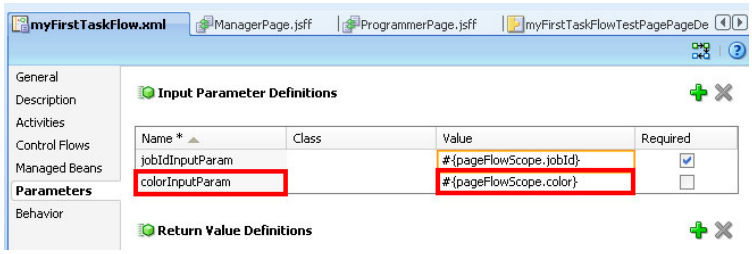
k) Now run the testpage and play around with the buttons and radiogroup.

2. *Bonus: Adding a Method Call*

One of the things you can do to add more functionality to your taskflow is adding a method call. We will add a method call to create a concatenation of the person's function and the person's favorite color. For that we need two things. First of all we need the possibility to enter a color into the taskflow. Second a method that executes the concatenation.

Let's start with the color. Notice that the test page already contains a dropdown to select a color. We need to assign the value of this dropdown to an input parameter of our taskflow.

a) Create a new input parameter for the taskflow.



b) In the test page assign the value of the dropdown to the input parameter. HINT: you can copy the jobId parameter that is already present in the page Definition, and adjust the values.

```
                version="11.1.1.55.36" id="myFirstTaskFlowTestPagePageDef"
                Package="nl.amis.technology.taskflow.handson.view.pageDefs"
<parameters/>
<executables>
  <variableIterator id="variables"/>
  <taskFlow id="myFirstTaskFlow1"
            taskFlowId="/WEB-INF/myFirstTaskFlow.xml#myFirstTaskFlow"
            activation="deferred"
            xmlns="http://xmlns.oracle.com/adf/controller/binding"
            Refresh="ifNeeded">
    <parameters>
      <parameter id="colorInputParam" value="#{TestBean.color}"/>
    </parameters>
  </taskFlow>
```

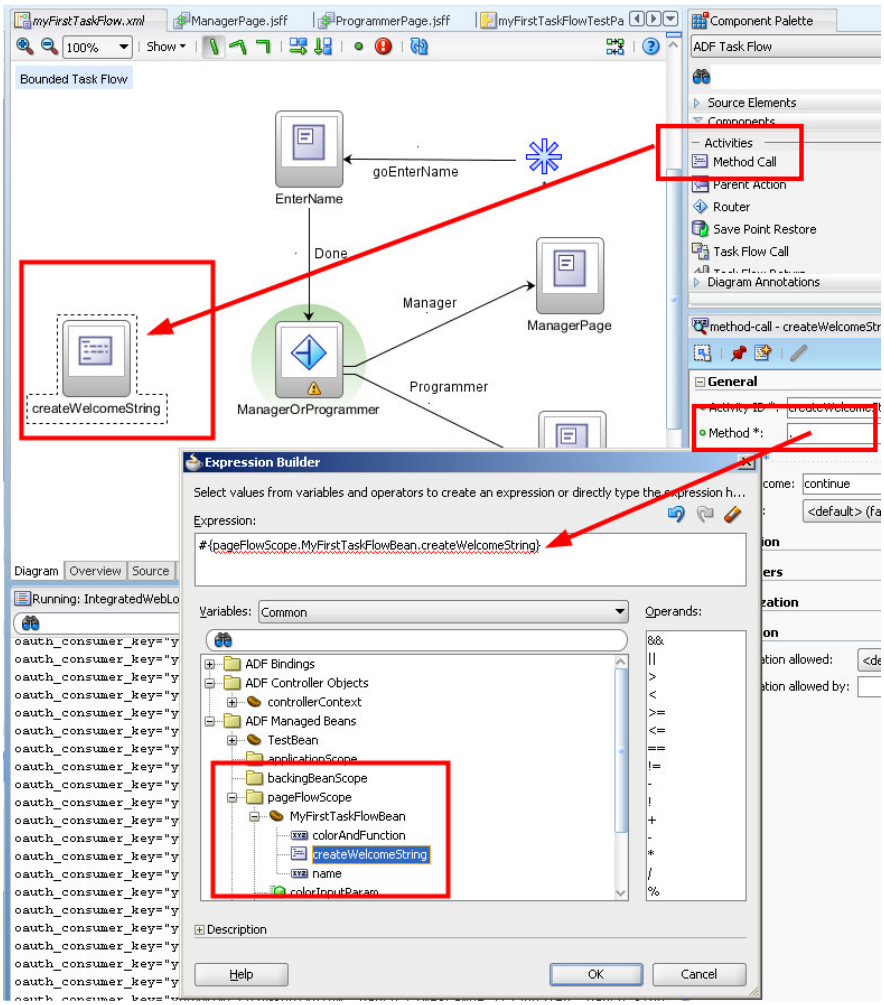
c) Now we need to create a Method that prints out the text:
"Hi there. You are a <function> and your favorite color is <color>". There is already a managed bean present in the taskflow. Open the class. First we create a bean property "colorAndFunction", and generate assessors for the property. Next we can easily create the method that can assemble our welcome text. Use two arguments for this method; one to hold the job and one to hold the color. You can copy the code of the method from the textbox below. The information we need is stored in the input Parameters. When we create the MethodCall activity, we have to provide these two parameters.

```
public void createWelcomeString(String job, String color) {

  if (job.equalsIgnoreCase("AD_PRES")) {
      job = "Manager";
  } else {
      job = "Programmer";
  }
  if (color == null) {
      setColorAndFunction("Hi there. You are a " + job +
                          " and you did not enter your favorite color.");
    } else {
      setColorAndFunction("Hi there. You are a " + job +
                          " and your favorite color is " + color);
    }
}
```

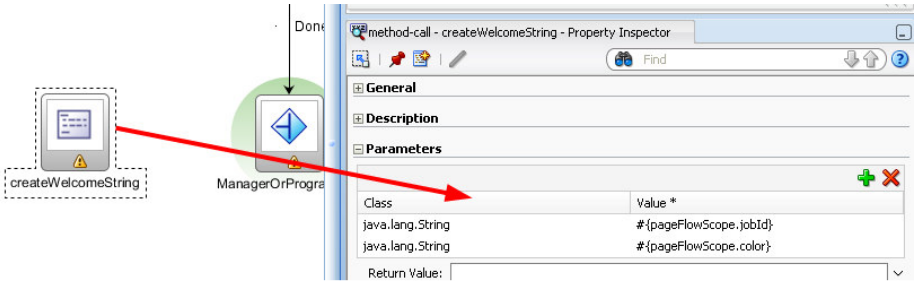
d) Add an outputText to the managerPage and programmersPage so you can see the text created by the method. Retreive the value form the colorAndFunction property.

```
<?xml version='1.0' encoding='UTF-8'?>
<jsp:root xmlns:jsp="http://java.sun.com/JSP/Page" version="2.1"
          xmlns:af="http://xmlns.oracle.com/adf/faces/rich">
 <af:panelGroupLayout id="pgl0" layout="vertical" >
  <af:outputText value="Hi, I Am the Programmer Page Fragment" id="ot1"  />
  <af:spacer height="10" id="s2"/>
  <af:commandButton text="Enter / Change your name" action="goEnterName"
                    id="cb1" />
  <af:spacer height="10" id="s1"/>
<af:panelGroupLayout layout="horizontal" id="pgl1">
  <af:outputText value="The name you entered is:" id="pam1" inlineStyle="color:Black;"/>
   <af:outputText value="#{pageFlowScope.MyFirstTaskFlowBean.name}" id="ot2"/>
</af:panelGroupLayout>
 <af:outputText value="#{pageFlowScope.MyFirstTaskFlowBean.colorAndFunction}"
                id="ot3"/>
</jsp:root>
```

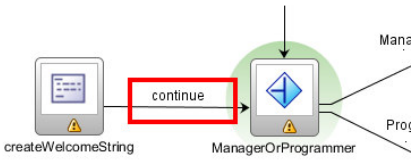
e) On the taskflow diagram create a method call activity "createWelcomeString" and set its method property to point to the method we just created.



f) The "createWelcomeString" method expects two arguments. We can provide these as parameters in the methodCall activity. Make sure to also provide the Class here. Leaving this empty will default to java.lang.Object which is not good at all. We need java.lang.String.

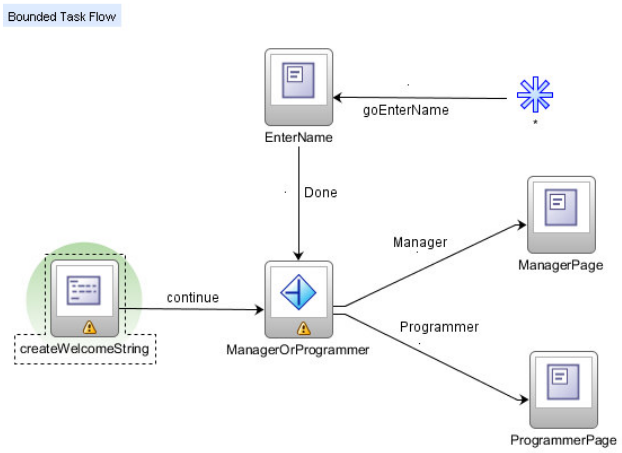


g) Set the fixed outcome property of the method call activity to "continue", and drag a "Control Flow Case" from the method call activity to the router. Set the "From Outcome" property of this flow case to "continue".



h) Final thing we need to do is to change the default activity of the taskflow to be the method call activity. Change this by invoking the rmb menu on the method call, and mark the activity as "default activity".

Notice the green circle has moved to the createWelcomeString.



Run the test page again. The taskflow still works but only starts at a different activity.