Ode to the taskflow

Hands-on Part II

Authors
Luc Bors & Lucas Jellema

Creation date
5 May 2010

Modification date
6 May 2010

Version/ status
1.0

Filename
hands on part II.docx

Document management

Revision history

date	author	version	remarks
5 May 2010	Lucas Jellema	0.9	
6 May 2010	Lucas Jellema	1.0	

Contents

1. My First ADF Library 1

1.1 Create the ADF Library for myFirstTaskFlow 1

1.2 Expose ADF Library as reusable asset in JDeveloper 7

1.3 Create Fusion Web Application that uses the ADF Library 9

1.4 Distributing updates in MyFirstTaskFlow 15

1.5 Bonus: Distributing your own flavor of MyFirstTaskFlow 16

2. Bonus: Absorbing some third party Taskflow 18

2.1 Consuming the UniversalConverter Taskflow 18

No table of figures entries found.

1. My First ADF Library

One of the main objectives for using Taskflows is the potential for reuse they offer. Taskflows can be developed as stand-alone, self-contained units that can easily be reused across the application - by binding the taskflows through multiple region-bindings in multiple pages. And that is not all: task flows can also be reused across multiple applications! We could create a library of reusable taskflows and make those available to all ADF projects in our organization. We could also create really useful, generic taskflows and publish them (or even sell them) to a worldwide audience. In fact, that is exactly what Oracle itself is doing through the WebCenter Services product.

In this second part of the Ode to the Taskflow hands-on lab, we will take a close look at the mechanism we need to use to achieve this reuse across applications. At the heart of it is the ADF Library - the bundle in which we publish our reusable taskflows with all their dependent objects. The ADF Library can subsequently be passed around and imported into other ADF applications. Once that is done, the taskflow can be added to pages in those other ADF applications, in the same way local taskflows are.

1.1 Create the ADF Library for myFirstTaskFlow

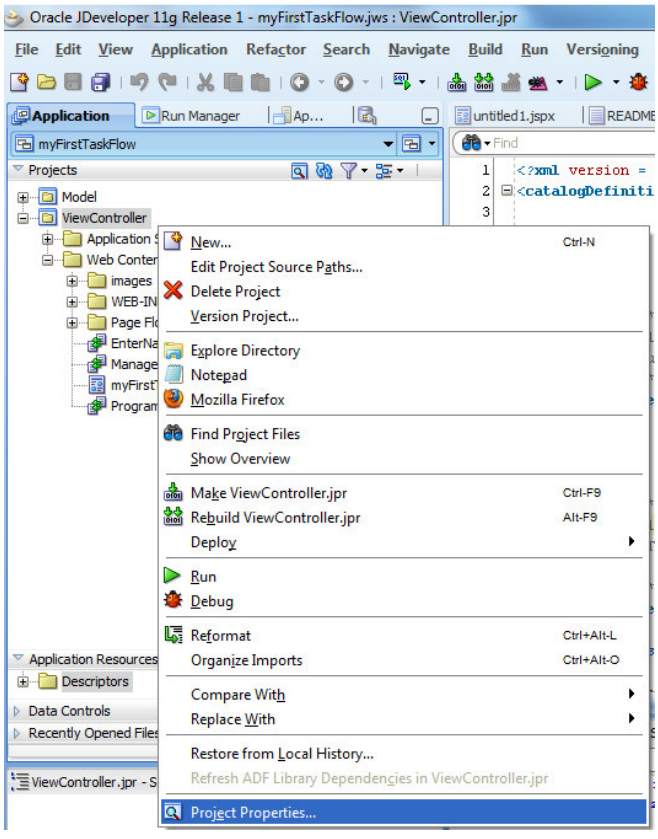
1. Open the Fusion Web Application myFirstTaskFlow. Either use the one you created in the previous lab, or start with the complete solution that was given to you in the end-myFirstTaskFlow.zip archive.

Create a central directory where we can store all reusable ADF libraries that all applications may share:

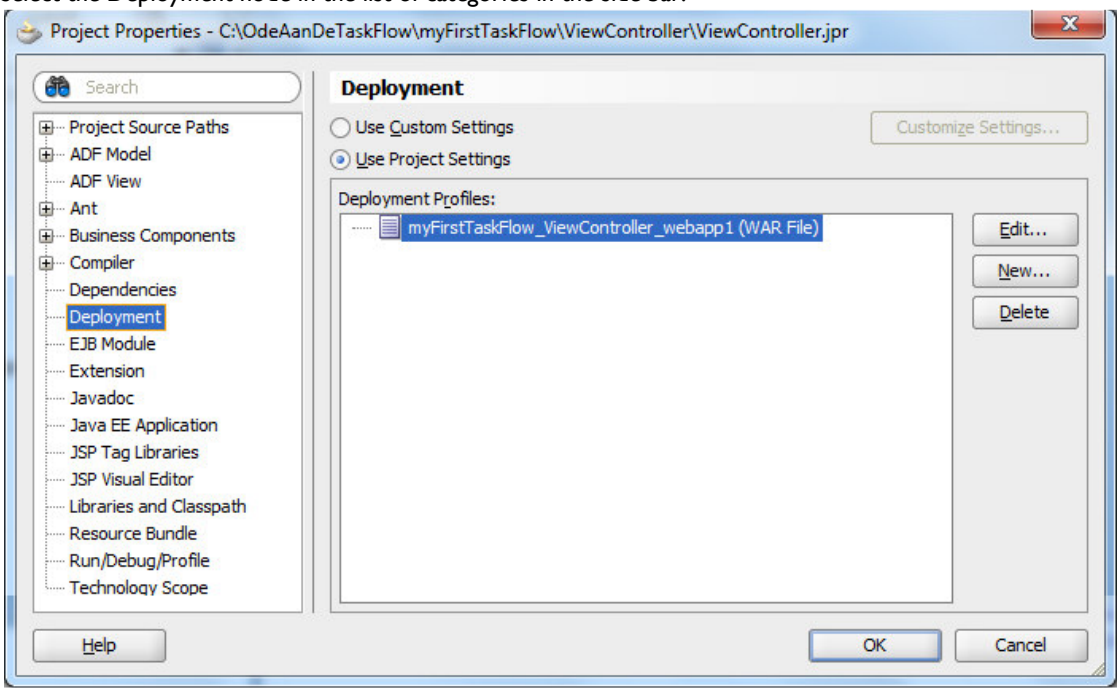
2. Create a directory on your local file system, called C:\ADF_LIBRARIES.

Create a new deployment profile – of type ADF Library - that we will use later on to bundle up and deploy the myFirstTaskFlow application.

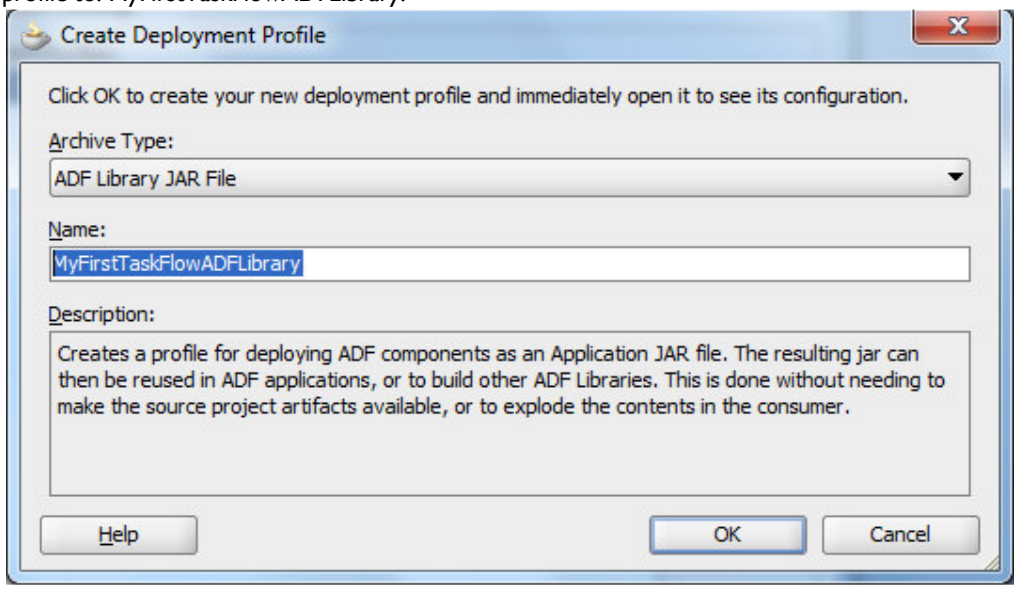
3. Right click the ViewController project and select Project Properties from the context menu.



Select the Deployment node in the list of categories in the side bar:

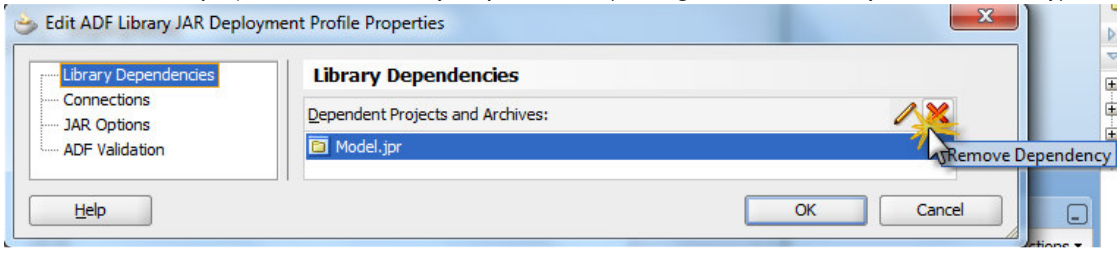


Click on New to create a new deployment profile. Select type ADF Library. Set the name of the deployment profile to: MyFirstTaskFlowADFLibrary.

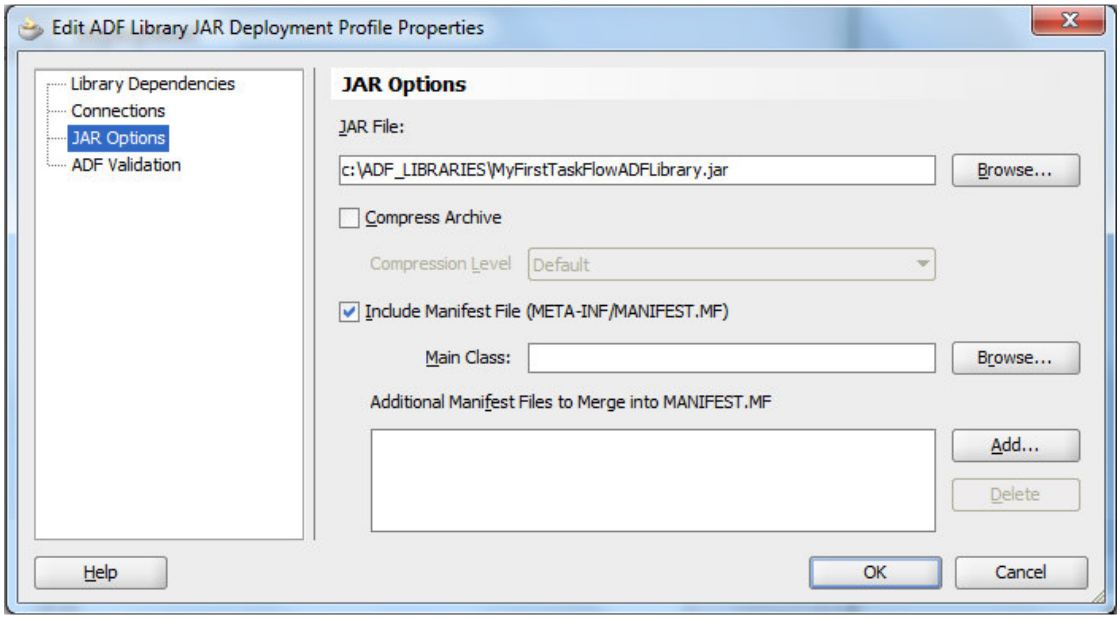


Click on OK.

Remove the Model project from the Library Dependencies (although it does not really hurt to let it stay).

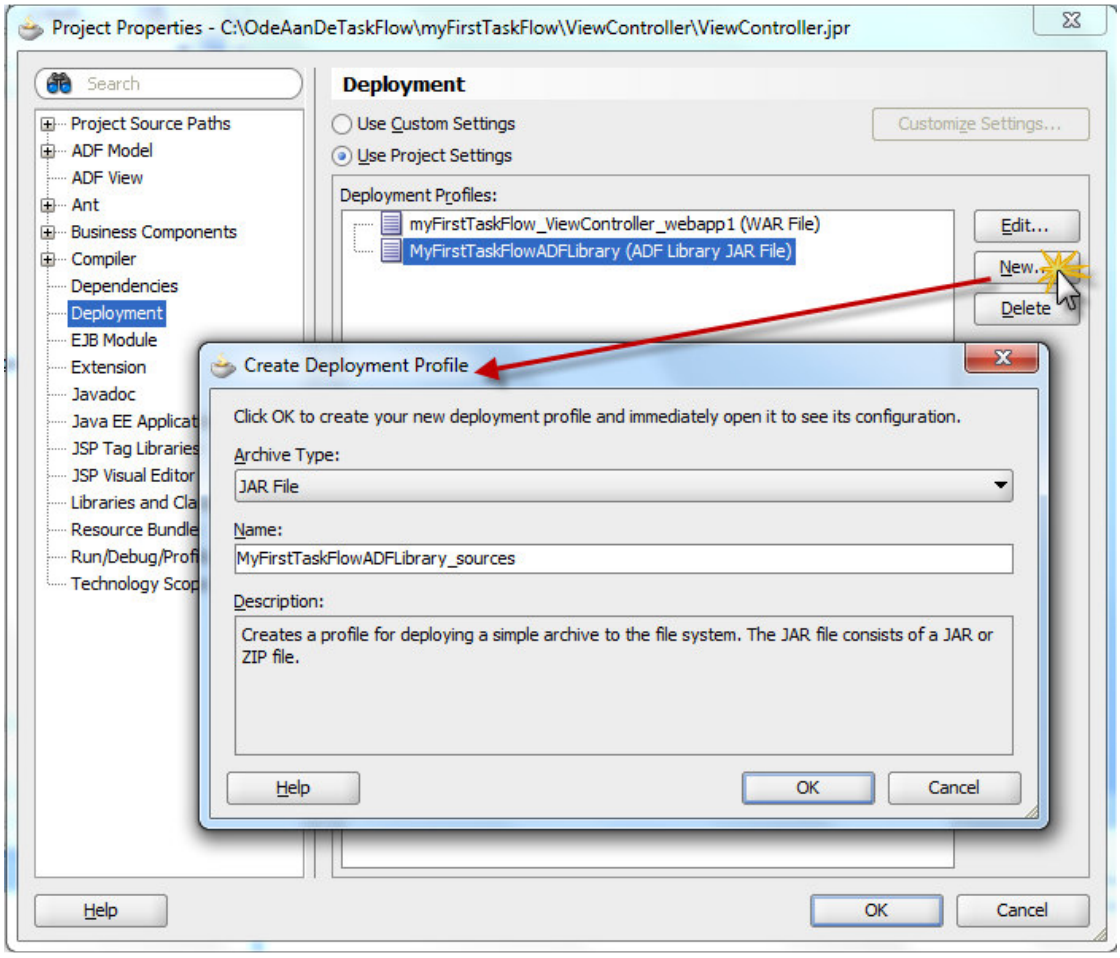


Click on JAR options in the side bar:



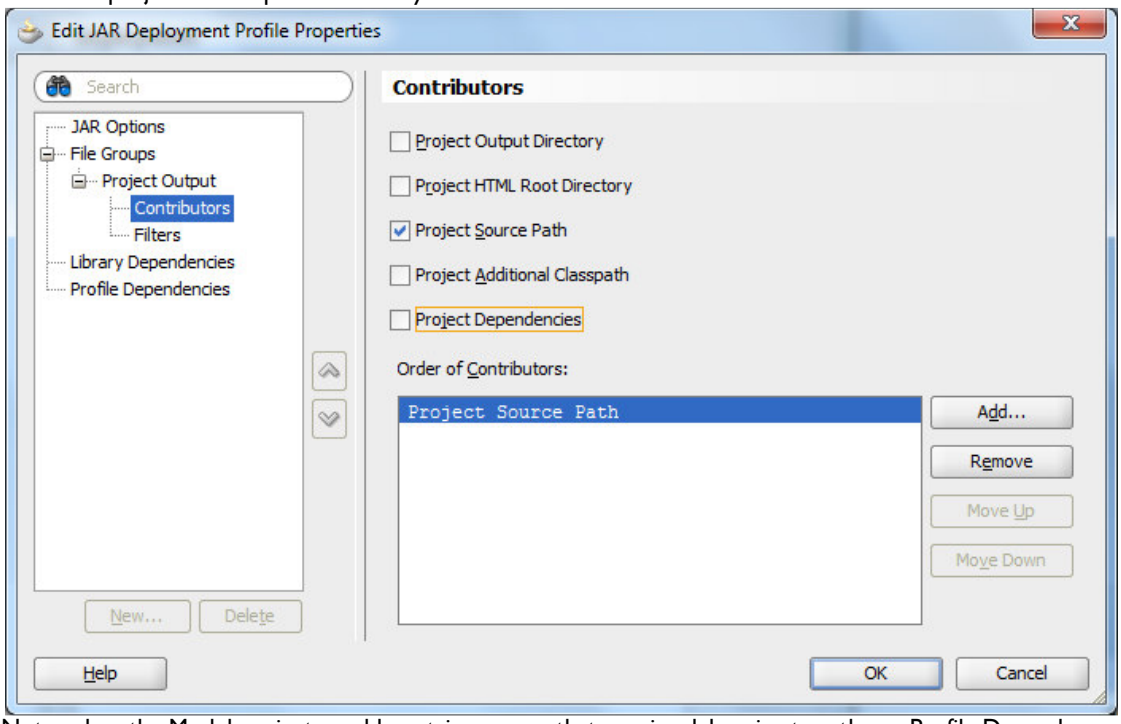
Set the name and location of the JAR file to c:\ADF_LIBRARIES\MyFirstTaskFlowADFLibrary.jar.

7. **Bonus**:
If we want to be able to debug through the code in the reusable taskflow when we are running an application that consumes the taskflow, we should also publish this source code in separate JAR file. To do so, create another deployment profile – of type JAR – with only the *Project Source Path* as Contributor (as we only need sources in this jar file);



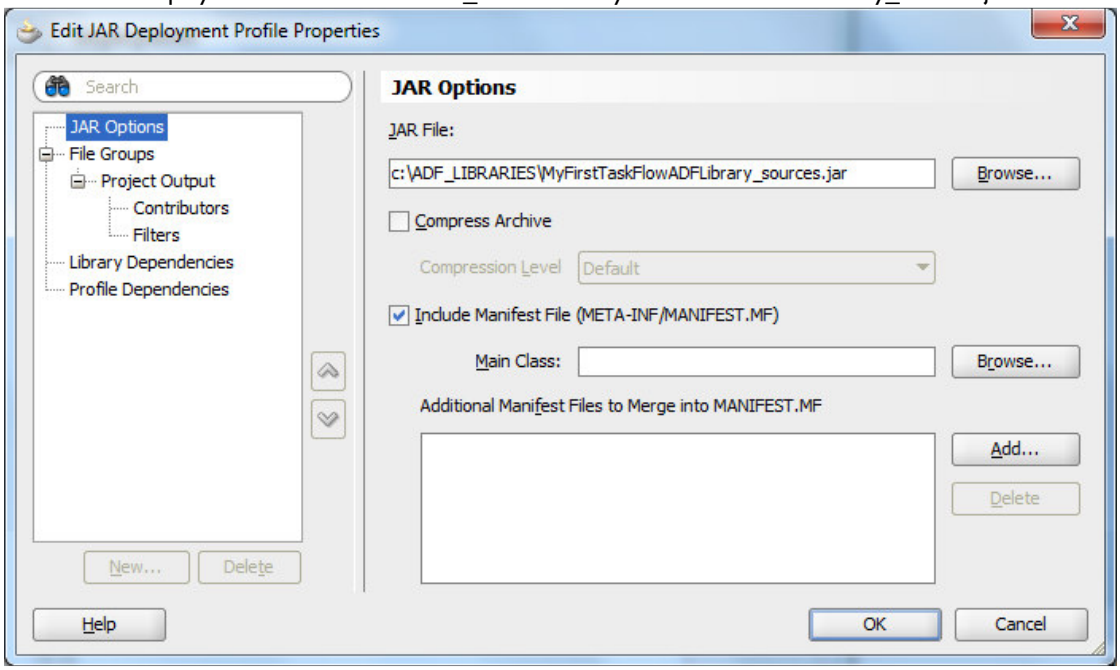
set the name of this archive to MyFirstTaskFlowADFLibrary_sources.

Select the project source path as the only contributor to this archive:



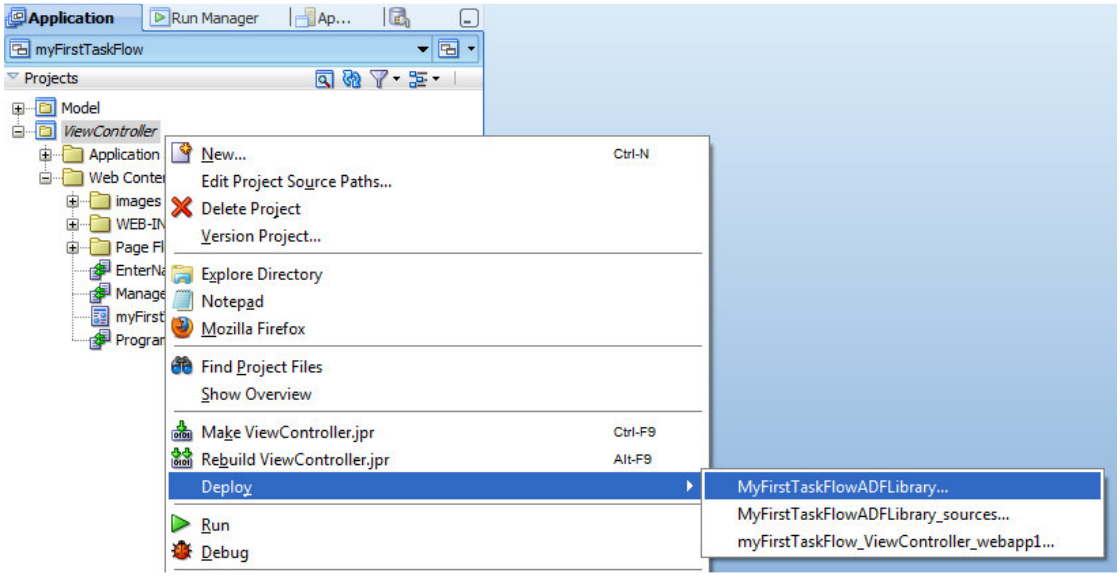
Note: when the Model project would contain sources that require debugging too, then a Profile Dependency on the Model project would have to be added.

Set name and deployment location to: c:\ADF_LIBRARIES\MyFirstTaskFlowADFLibrary_sources.jar

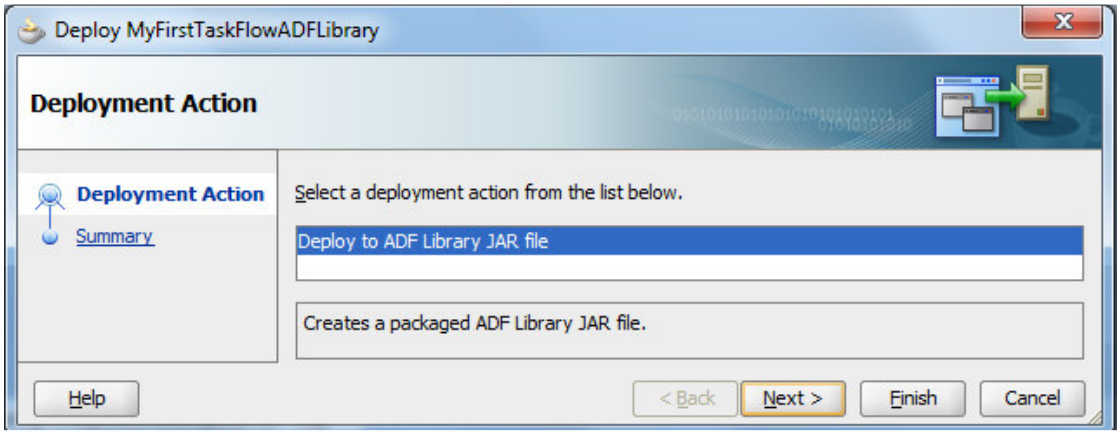


Deploy according to the deployment profile to create the ADF Library jar-file.

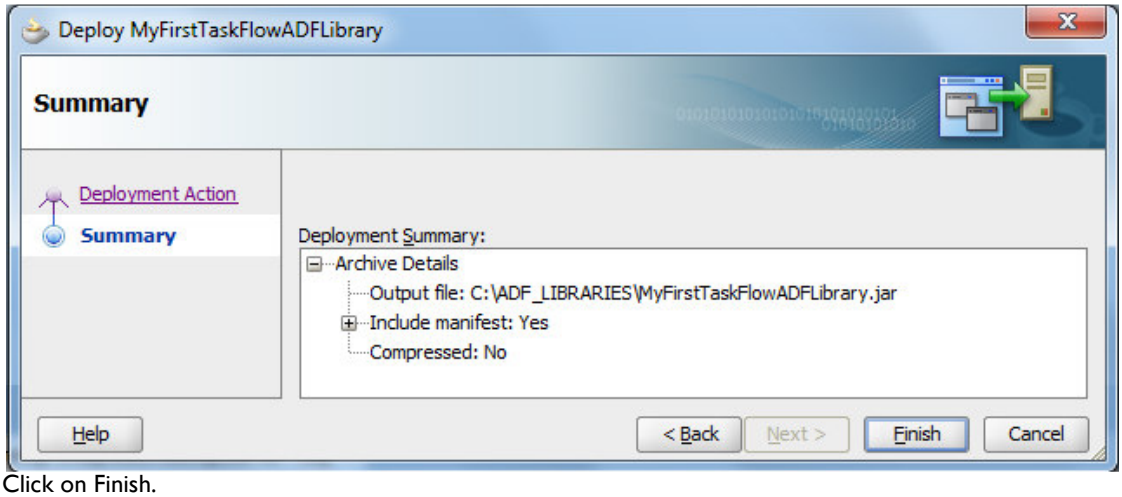
8. Right click the ViewController project node. Select Deploy in the context menu and next select the MyFirstTaskFlowADFLibrary deployment profile.



The deployment wizard appears:

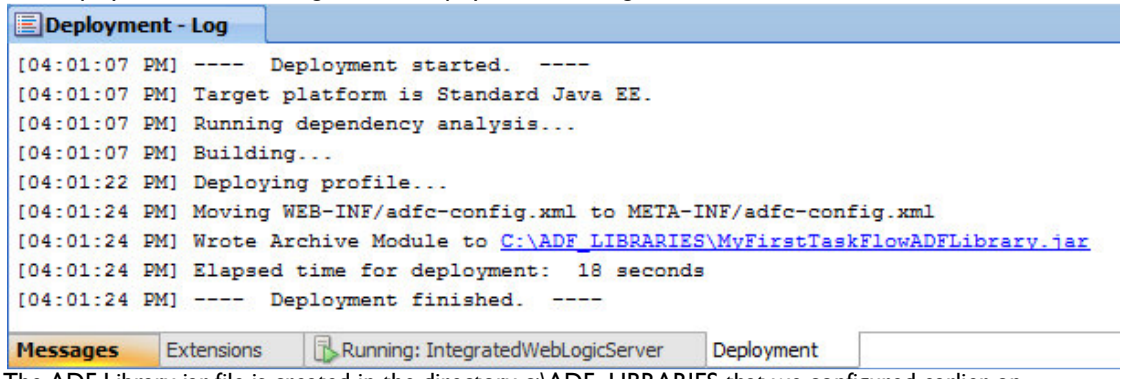


Click on Next. This brings up the summary page.



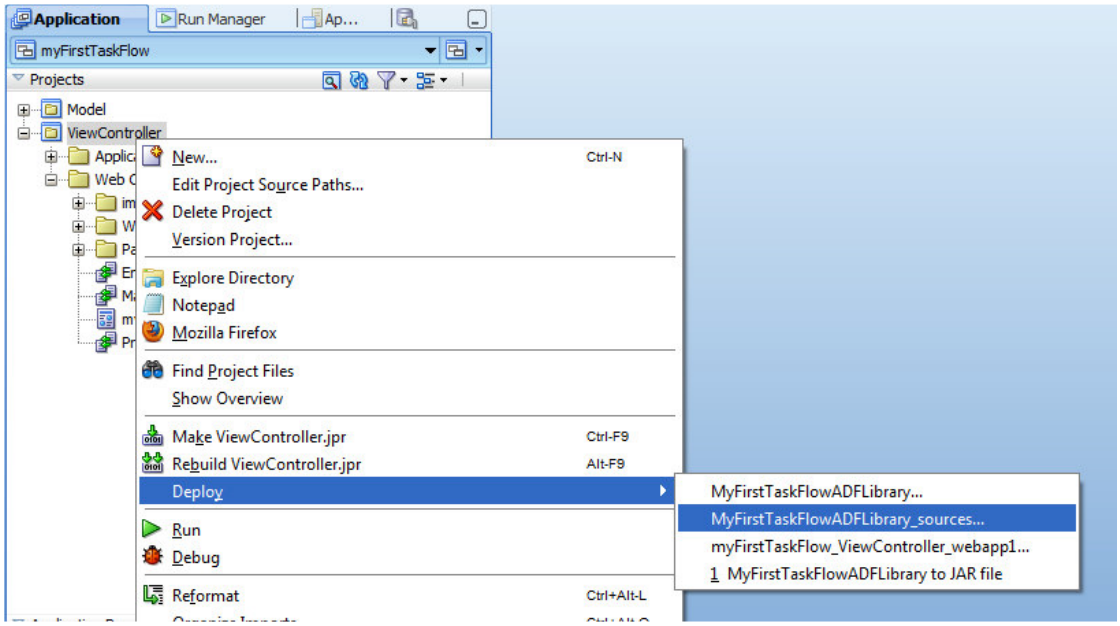
Click on Finish.

The Deployment tab in the Log window displays the following feedback:

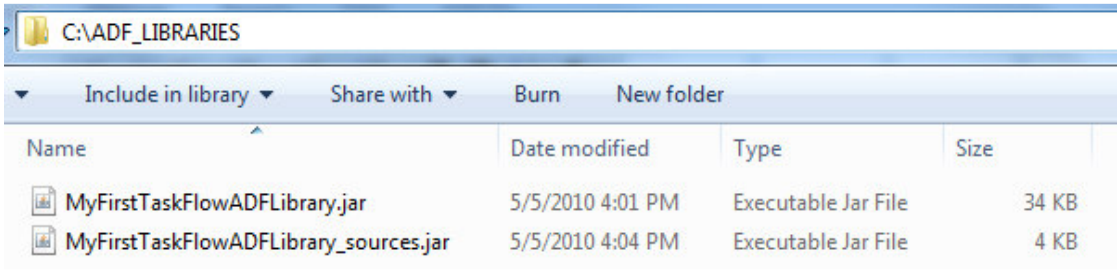


The ADF Library jar file is created in the directory c:\ADF_LIBRARIES that we configured earlier on.

9. Bonus: if you want to be able to debug through all the sources belonging to the ADF Library, than also deploy the ViewController project according to the MyFirstTaskFlowADFLibrary_sources deployment profile:



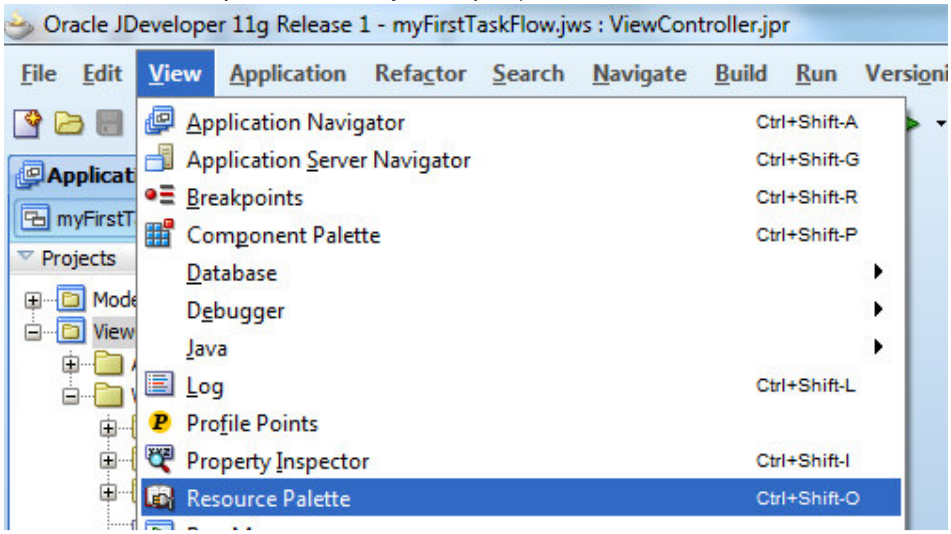
The same deployment wizard appears. Click Next and Finish. The jar file is created into the c:\ADF_LIBRARIES directory.



1.2 Expose ADF Library as reusable asset in JDeveloper

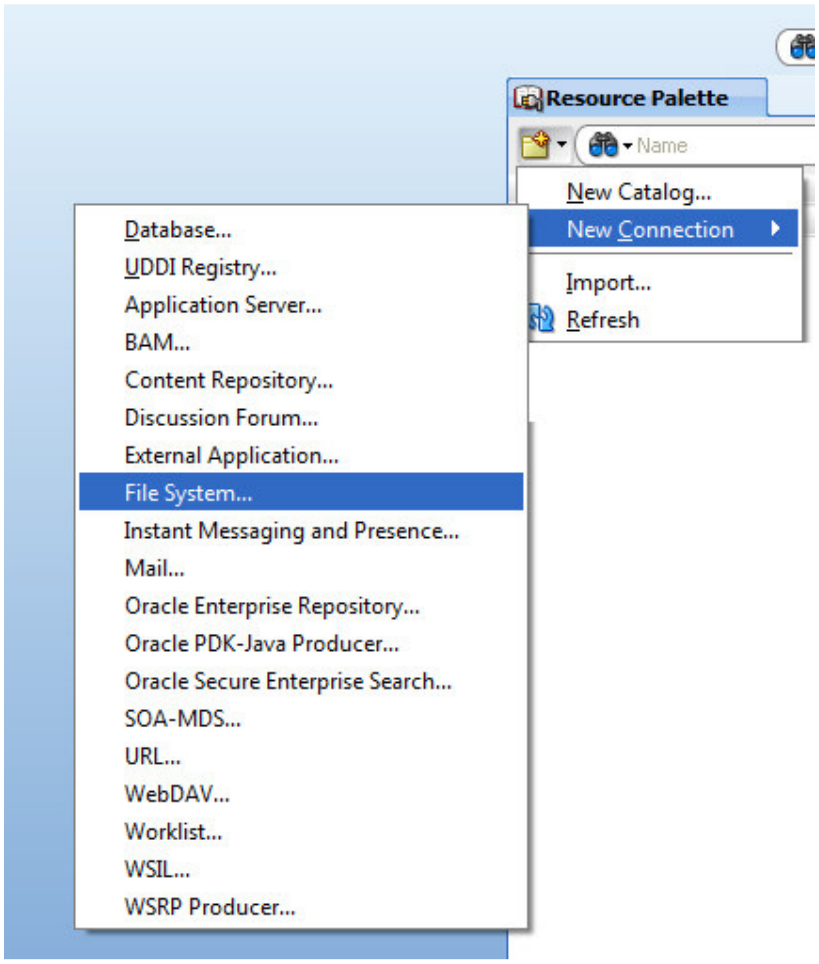
Steps:

1. Make the resource palette visible in JDeveloper (choose Resource Palette from the View menu)

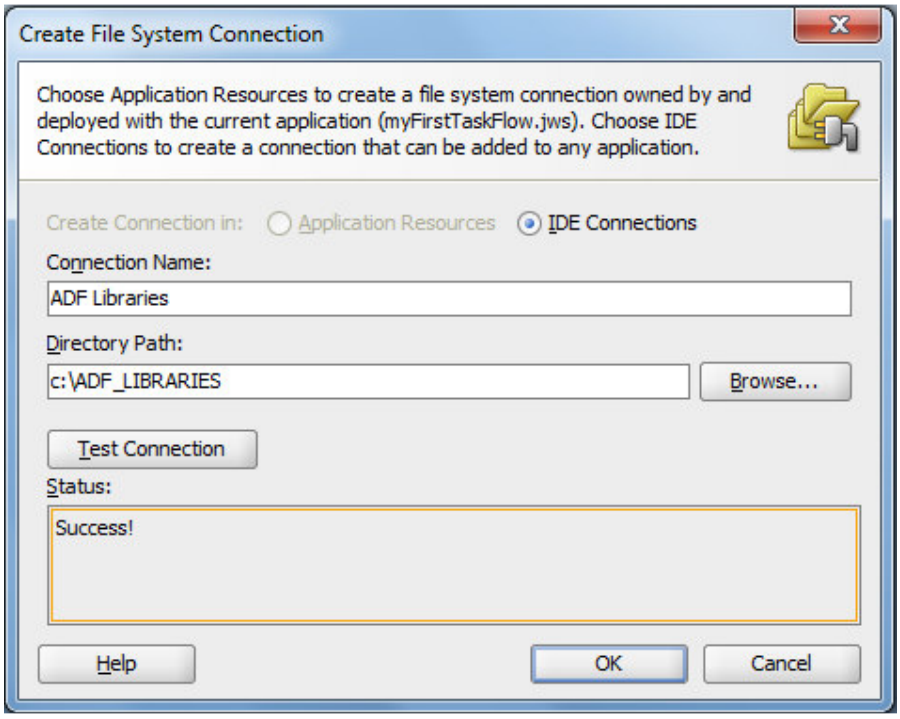


Create a new File System connection in the Resource Palette; link this connection to the directory c:\ADF_LIBRARIES where you deployed the ADF Library to. This will make the ADF Libraries in this directory available in the IDE and allows the developer to add task flows from the ADF Library to the application she is working on.

2. Open the drop down on the Resource Palette. Choose New Connection and the option File System from the child menu.

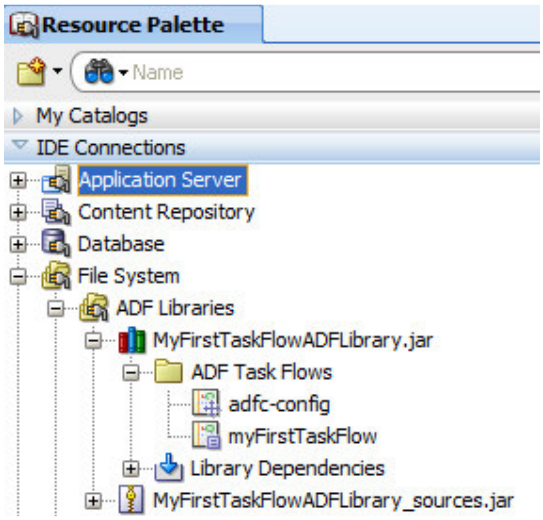


The Create File System Connection pops up. Enter ADF Libraries as the name for this connection. Specify c:\ADF_LIBRARIES as the Directory Path. Click on Success to verify whether the connection can be created successfully.



Click OK to complete the creation of the connection.

3. The Resource palette should now show the MyFirstTaskFlowADFLibrary and the MyFirstTaskflow taskflow in it

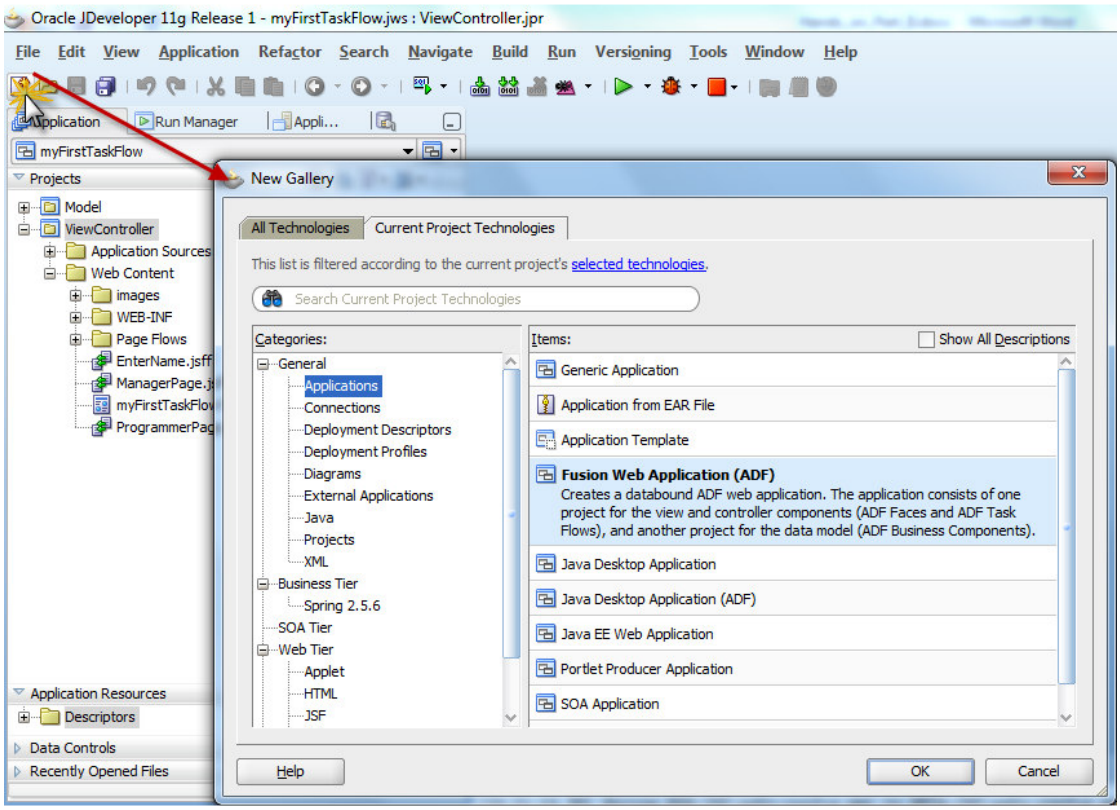


1.3 Create Fusion Web Application that uses the ADF Library

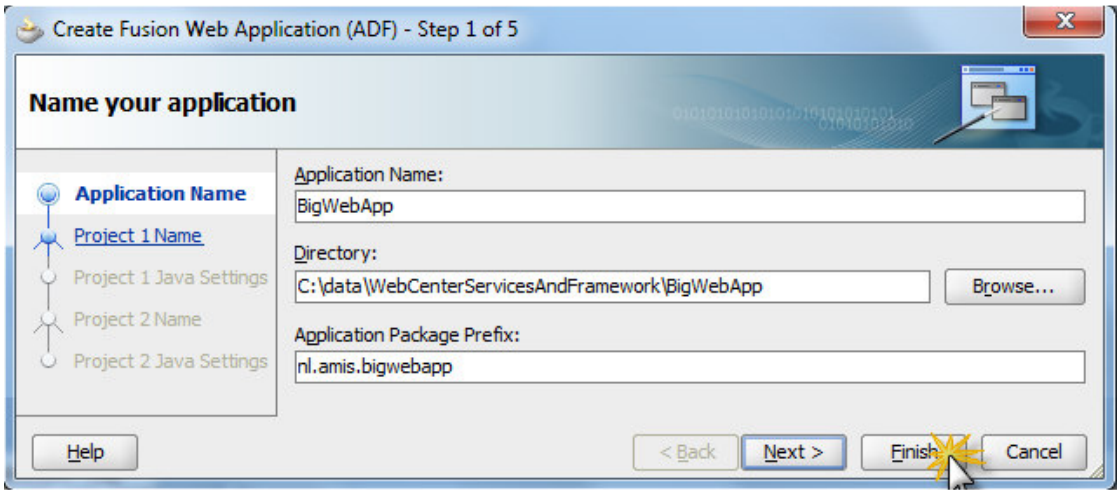
Steps:

Create New Fusion Web Application BigWebApp.

1. Click on the new icon to bring up the New Gallery. Select the category node General - Applications. Select the item Fusion Web Application (ADF). Click on OK.



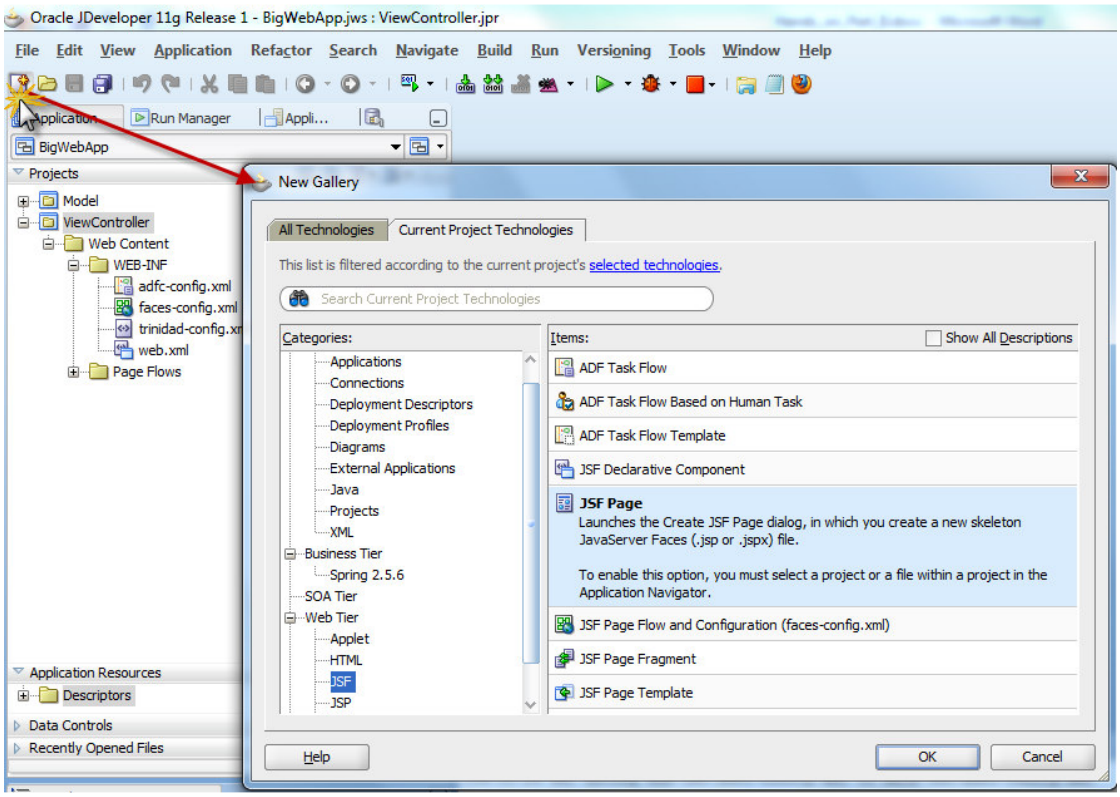
Click on OK.



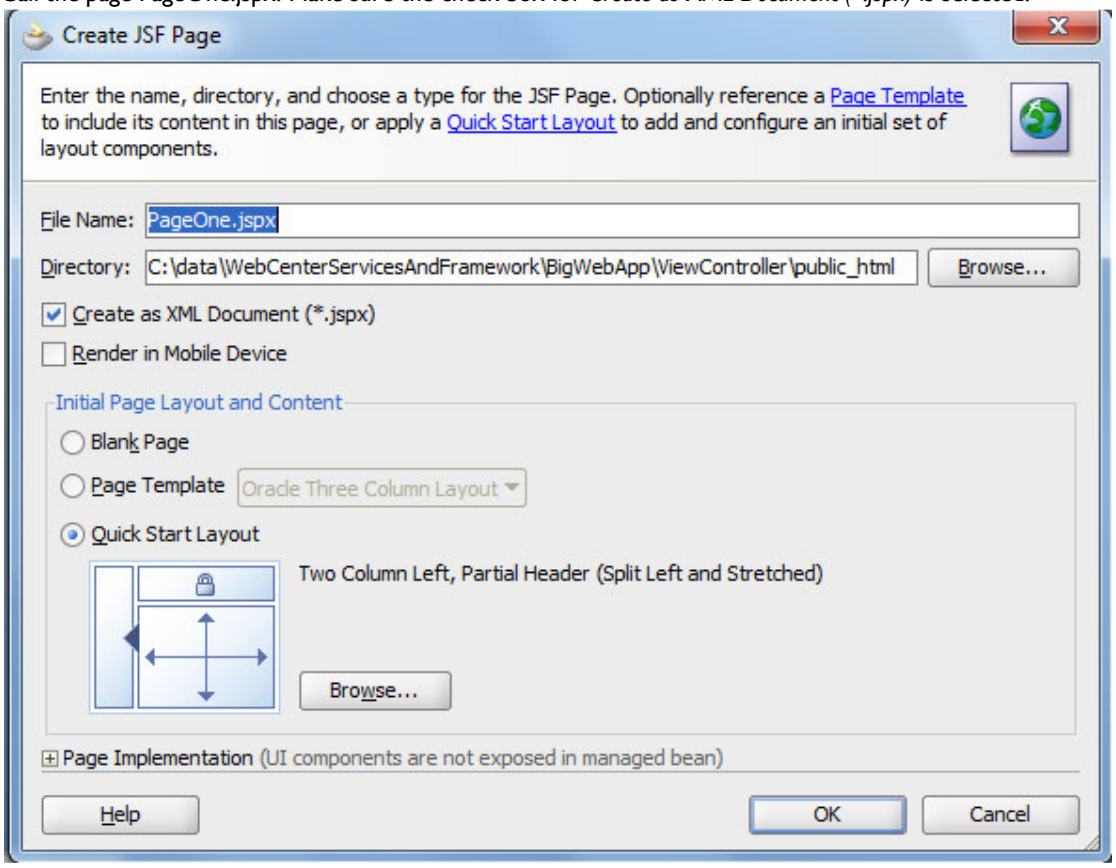
The Create Fusion Web Application (ADF) wizard appears. Type BigWebApp as the name of the new application. Set the Application Package Prefix to nl.amis.bigwebapp. Click on Finish to have the application with the Model and ViewControll projects created.

All subsequent steps are for the ViewController project:

2. Create a new JSF page



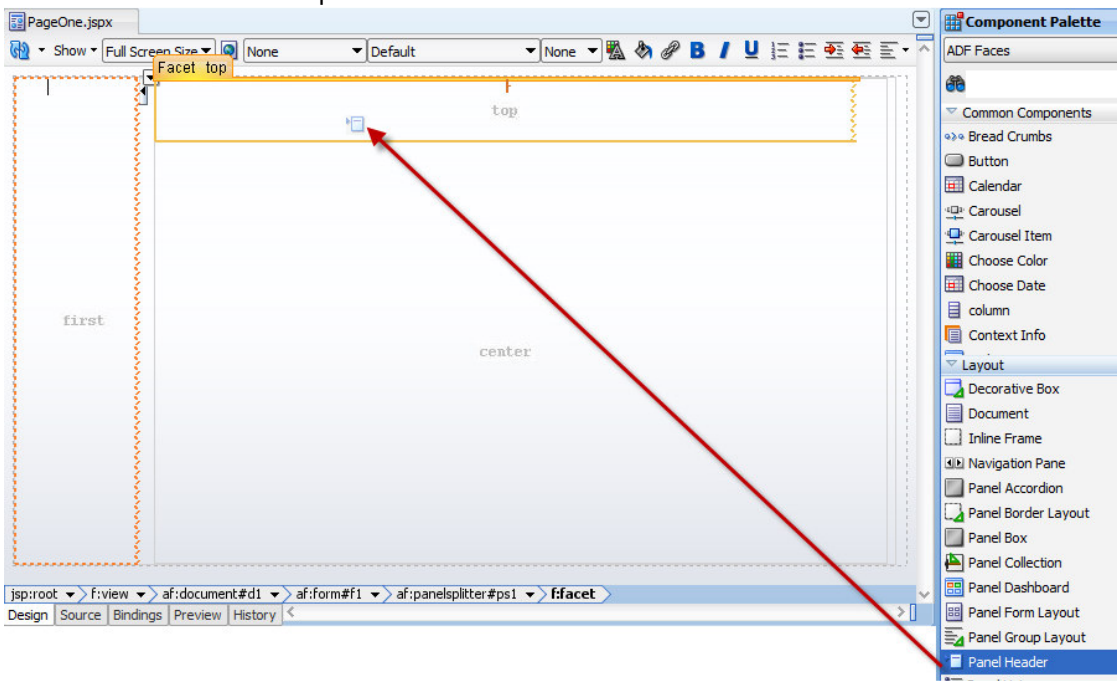
Call the page PageOne.jspx. Make sure the check box for *Create as XML Document (*.jspx)* is selected.



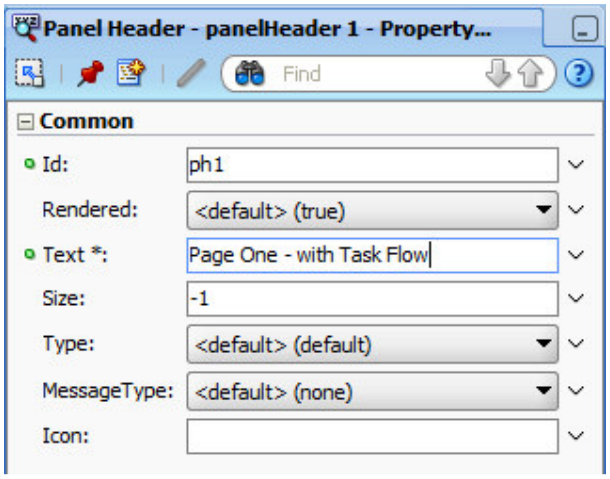
Click on OK.

The visual editor will open with this new page in it.

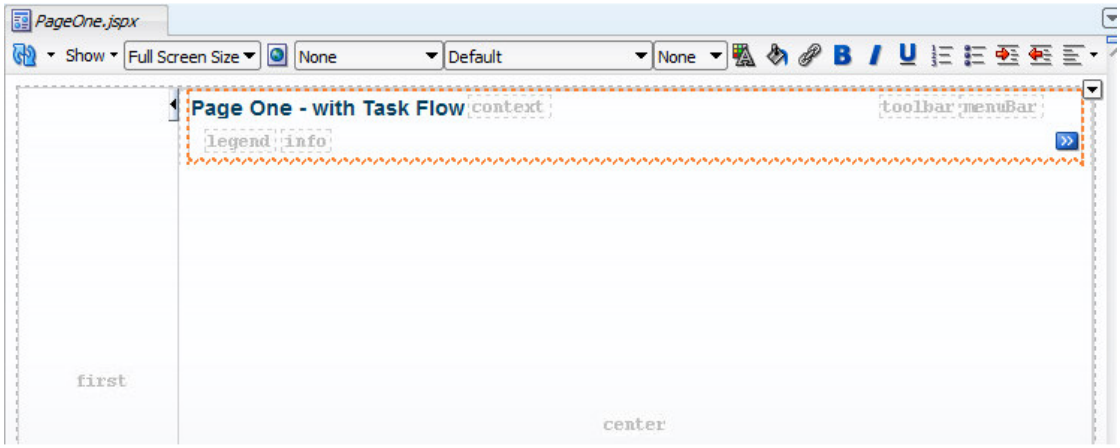
3. Add a PanelHeader to the top facet



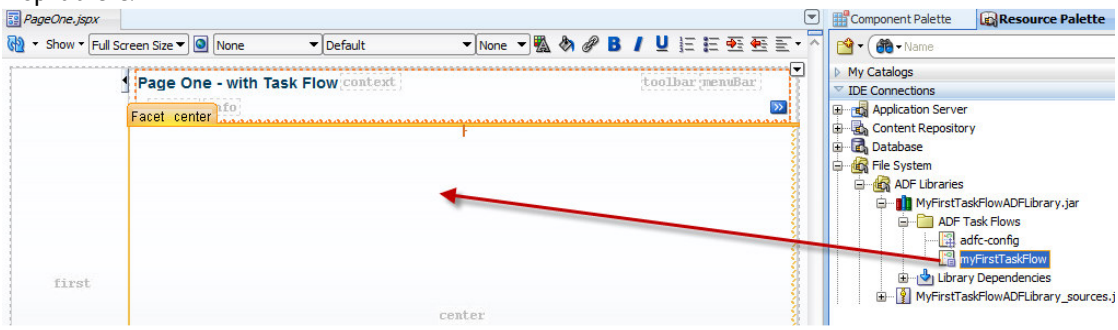
and set the text property to Page One - with Task Flow.



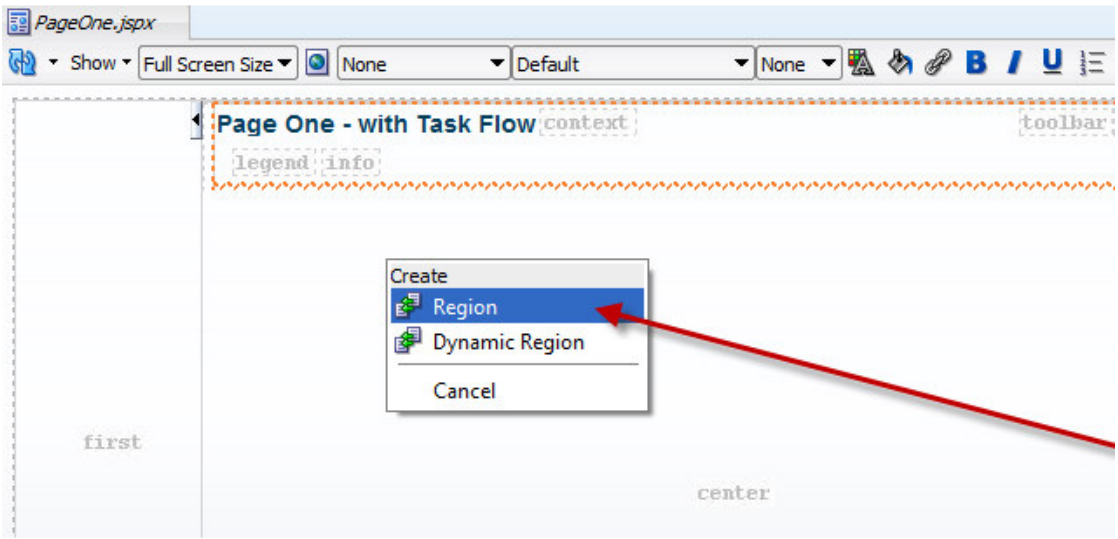
The visual editor reflects this change:



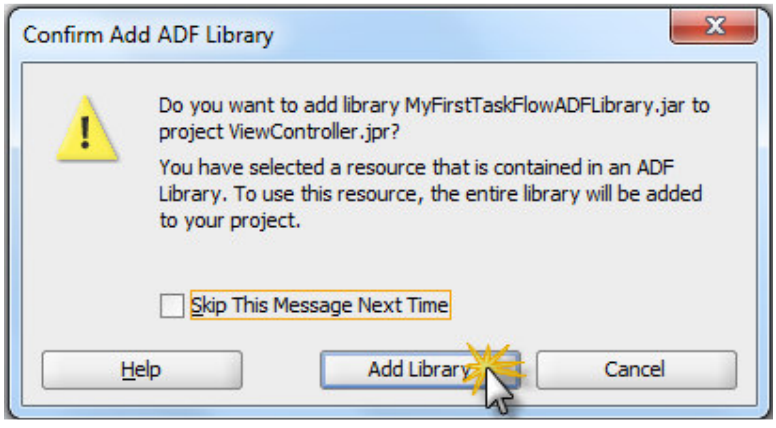
4. Drag the MyFirstTaskflow taskflow in the MyFirstTaskFlowADFLibrary to the center facet of the page and drop it there.



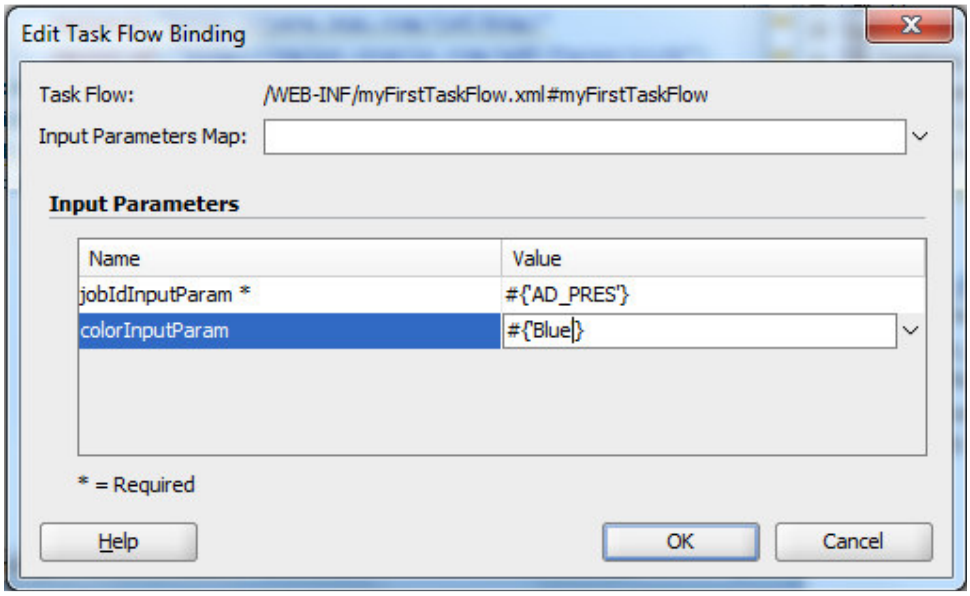
When you drop it, you have to select either Region or Dynamic Region. Select Region in this case.



After you selected Region, you will be asked by JDeveloper whether to add the ADF Library to the project. Press Add Library to accept this.

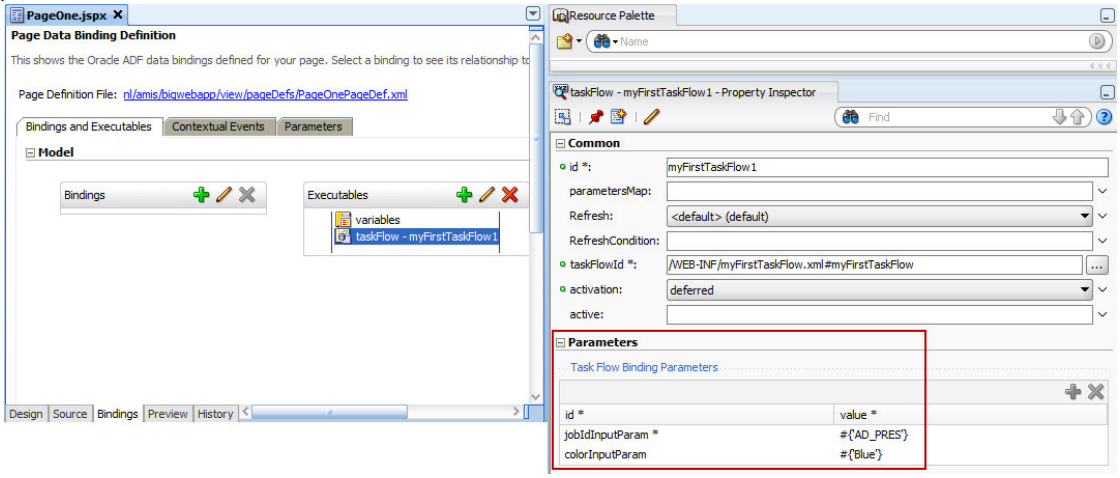


You need to provide values for the input parameters of the taskflow.

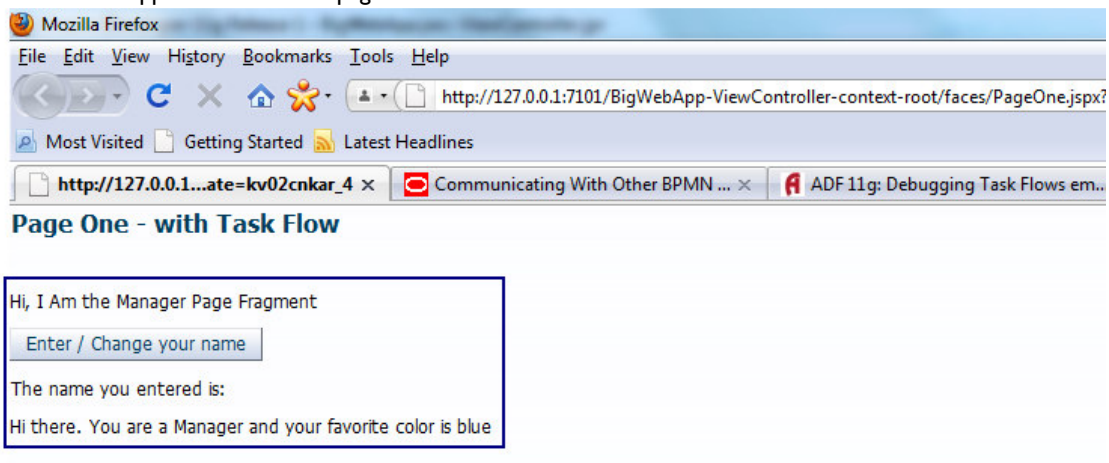


And click OK.

Note: you can always inspect and change the values of the input parameters of a task flow in the Bindings tab of the page; select the Region Binding that defines the TaskFlow usage. In the property palette, the input parameters are visible and their values can be modified.



5. Run the PageOne.jspx page to see whether we have indeed been able to embed the task flow functionality into this new application and new page.



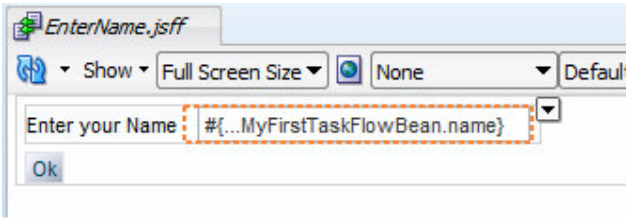
The rectangle indicates which section of the page is included from the reusable taskflow.

1.4 Distributing updates in MyFirstTaskFlow

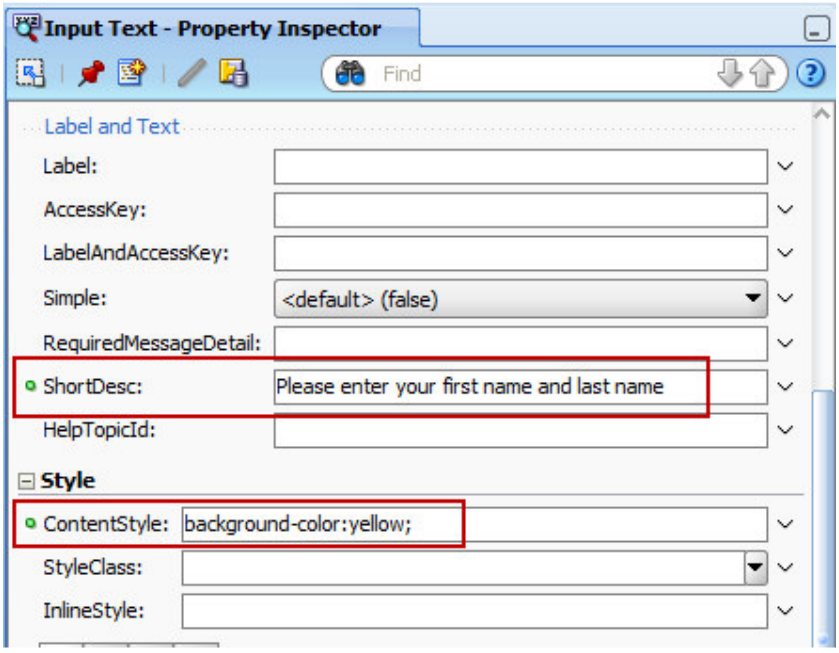
We want to see how we make changes in the taskflow and have those changes propagated to the applications that make use of it.

Return to the application myFirstTaskFlow.

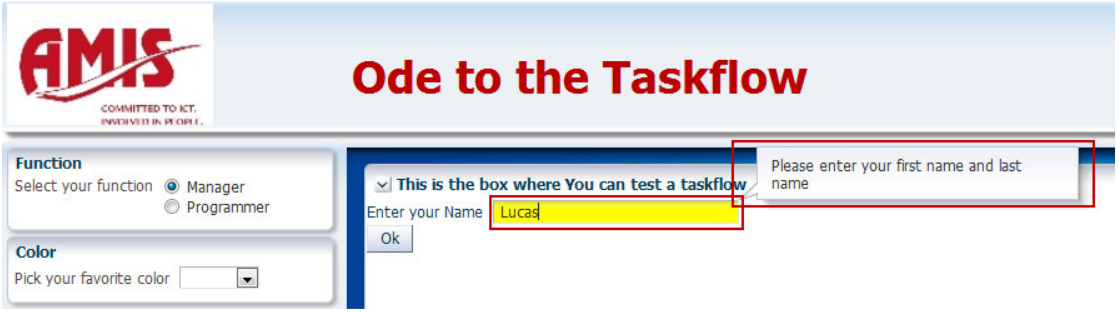
1. Open the EnterName.jsff fragment. Select the inputText component.



2. In the property palette, set the ShortDesc property to *Please enter your first name and last name*. Set the ContentStyle to *background-color:yellow;*

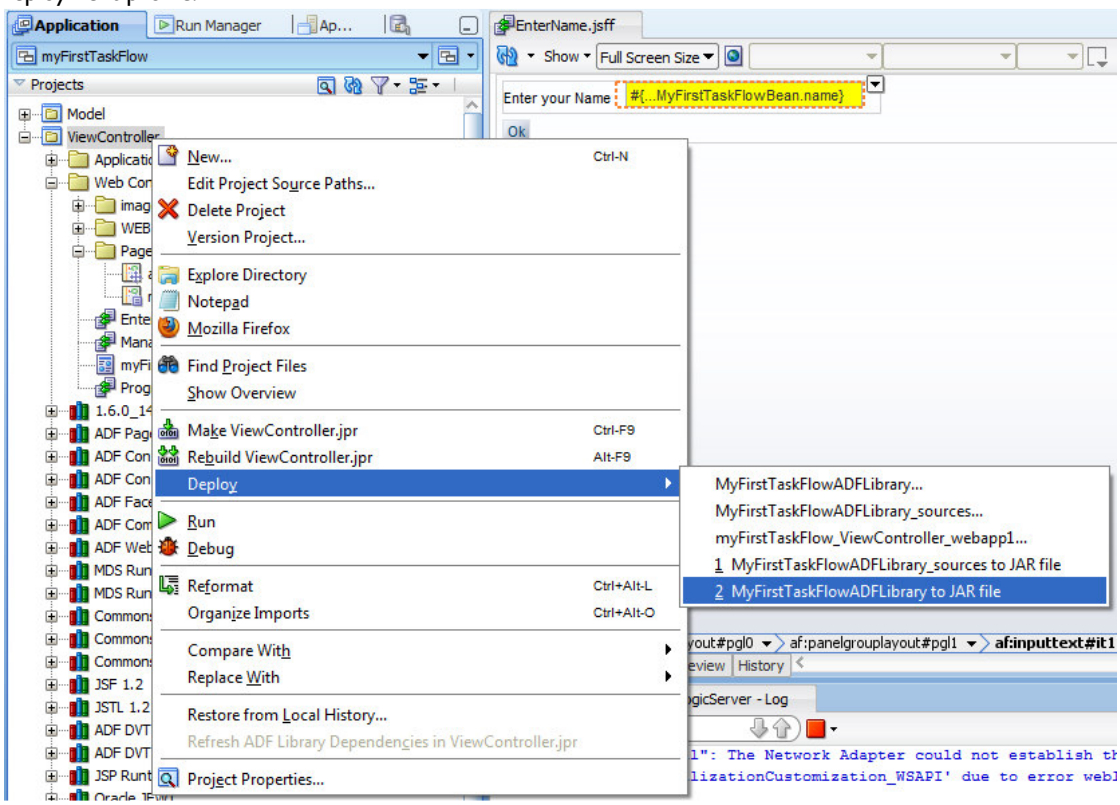


3. Run the page myFirstTaskFlowTestPage.jspx to see if the changes have been applied correctly.

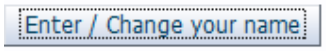


Our changes have been successfully applied to myFirstTaskFlow.

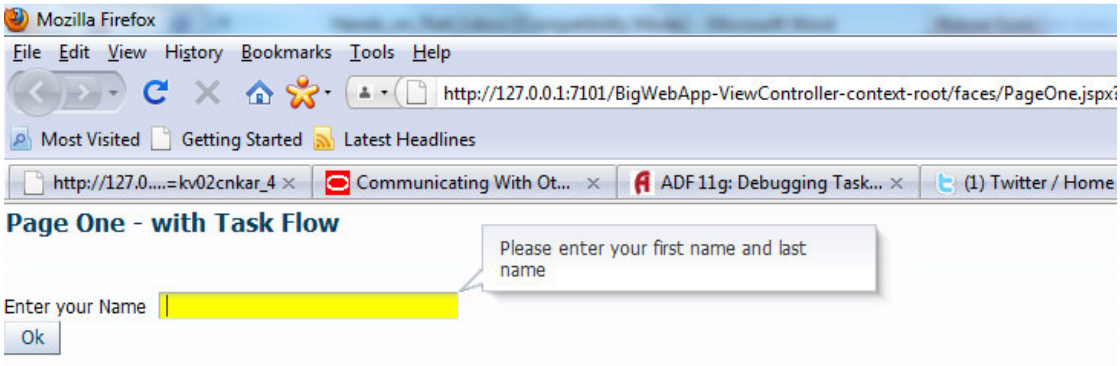
4. To distribute them, you need to redeploy the ViewController project, according to the ADF Library deployment profile:



5. Switch to the application BigWebApp. (re)run page PageOne.jspx.

When you press the button 

to enter or change the name, you should find that the taskflow's changes have propagated into the BigWebApp application.



1.5 Bonus: Distributing your own flavor of MyFirstTaskFlow

Return to the application myFirstTaskFlow. Open the page fragment ManagerPage.jsff.

1. Modify the layout of the page. Apply some of your own creativity. Change the text, add images or more boilerplate text. Change the styling. Add additional validations. Whatever you can think of to make it very much your page.

2. Redeploy the ViewController project to the ADF Library jar file.

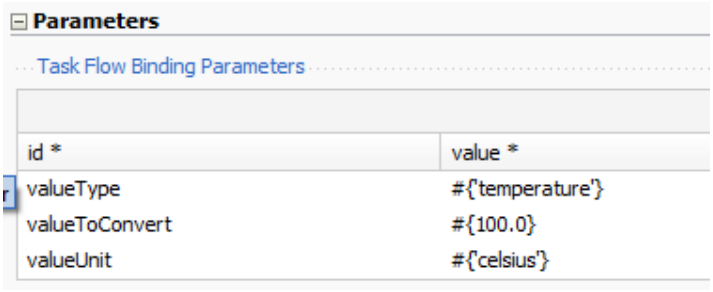
3. Hand your ADF Library over to your neighbor. Ask him or her to add your file to his or her c:\ADF_LIBRARIES directory and use your taskflow in her/his application.

4. Get a personal flavored ADF Library from your neighbor. Copy it to your c:\ADF_LIBRARIES directory, under a unique name. Refresh the file system connection in the resource palette. See the new file appear. Use the taskflow that was personalized by your 'dealer' in one of your application's pages.

2. Bonus: Absorbing some third party Taskflow

You will be given an ADF Library file called UniversalConverterADFLibrary.jar. This jar file contains a single ADF Taskflow that helps with the conversion of temperatures.

It takes three input parameters:



the valueType has to be set to #{'temperature'}. The valueToConvert has to be an EL expression that evaluates to a Float value (for example 23.6, 98.4, 0.0). The valueUnit must be a string; supported values are celsius, fahrenheit, kelvin and rankine.

2.1 Consuming the UniversalConverter Taskflow

1. See whether you can get this taskflow up and running in your application.

2. One way of using this converter could be inside a popup that appears when an inputText is manipulated that contains a temperature value.

Something like:

```
                <af:inputText label="Temperature" id="it1" >
                            <af:showPopupBehavior popupId="tempPU"
                                                  triggerType="contextMenu"
                                                  align="startAfter"
                                                  alignId="it1"/>
                 </af:inputText>
```

and

```
        <af:popup id="tempPU">
          <af:panelWindow id="pw" title="Temperature Conversion">
            <af:region
value="#{bindings.universalconvertertaskflow1.regionModel}"
                       id="r1" partialTriggers="::it1"/>
          </af:panelWindow>
        </af:popup>
```