ADF Mobile Code Corner

m06. How-to synchronize two input fields bound to a managed bean

twitter.com/adfcodecorner

Abstract:

ADF Mobile does not provide partial refresh properties for dependent fields to refresh and instead uses a JavaBean property change notification mechanism.

This article shows the most simplistic "non-hello-world" sample and demonstrates the steps you need to perform to create a form that contains two input fields bound to a managed bean. Changing the value of one field will copy the same to the second field and refresh the screen so the copied string displays properly.

Author: Frank Nimphius, Oracle Corporation
twitter.com/fnimphiu
16-APR-2013

Oracle ADF Mobile Code Corner is a blog-style series of how-to documents targeting at Oracle ADF Mobile that provide solutions to real world coding problems. ADF Mobile Code Corner is an extended offering to ADF Code Corner

Disclaimer: All samples are provided as is with no guarantee for future upgrades or error correction. No support can be given through Oracle customer support.

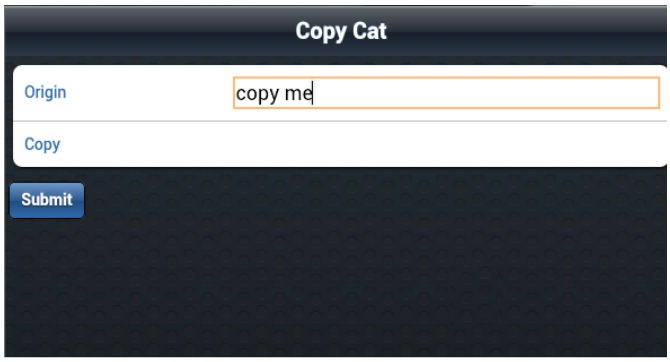
Please post questions or report problems related to the samples in this series on the OTN forum for Oracle JDeveloper: http://forums.oracle.com/forums/forum.jspa?forumID=83

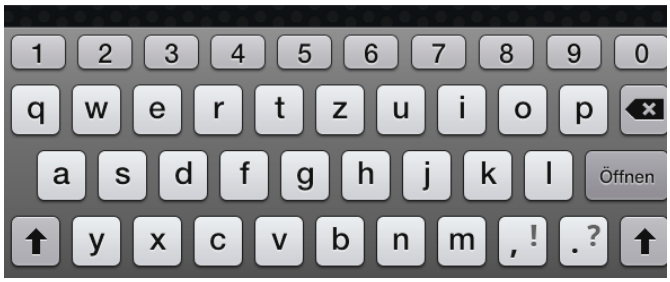
Introduction

As the ADF Mobile product documentation points it out: *"The ADF Mobile runtime will register itself as a listener on managed bean property change notifications so that EL expressions bound to UI components that reference bean properties will update automatically if the value of the property changes."* All code that is required for this to work is generated by Oracle JDeveloper, as shown in the following step-by-step tutorial.

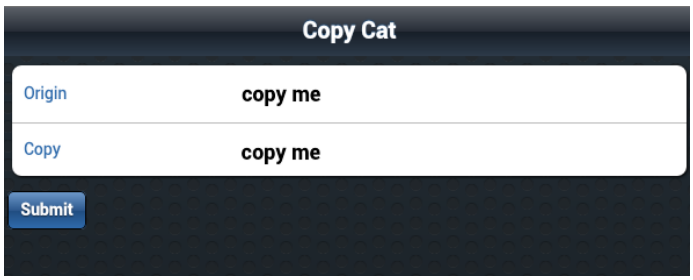
At the end, you should see a icon **CopyCat** on your mobile device to launch the simple sample application.

The form nelow shows an editable input field **Origin** and a read-only field **Copy**. Editing the **Origin** field and stepping out of it will update the value of the **Copy** field and refresh the screen so the data change can be displayed.





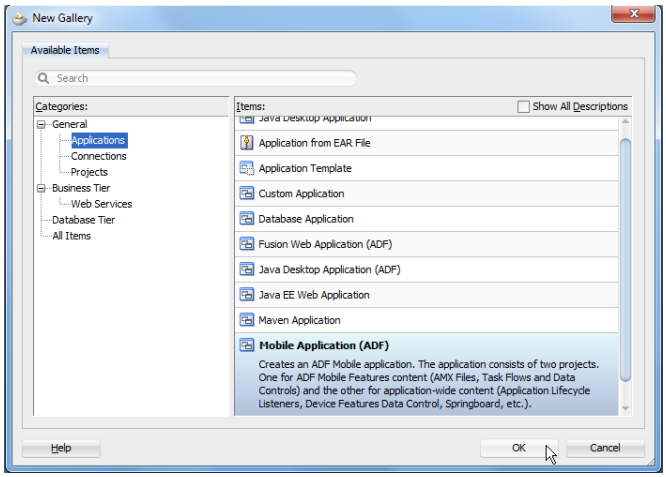
The image below shows the **CopyCat** sample in action.



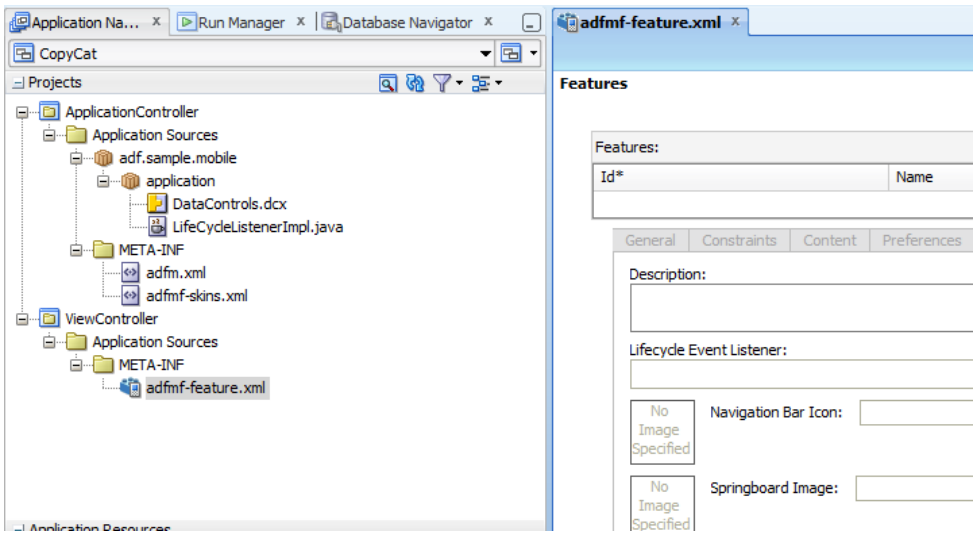
Building the Form

To build a mobile application, you require at least JDeveloper 11.1.2.3 (11g R2 PS2) with the ADF Mobile extension installed (Choose Help -> Check for Update in case you don't have it installed).

In Oracle JDeveloper, create a new application and choose the **Mobile Application (ADF)** template as shown in the next image.

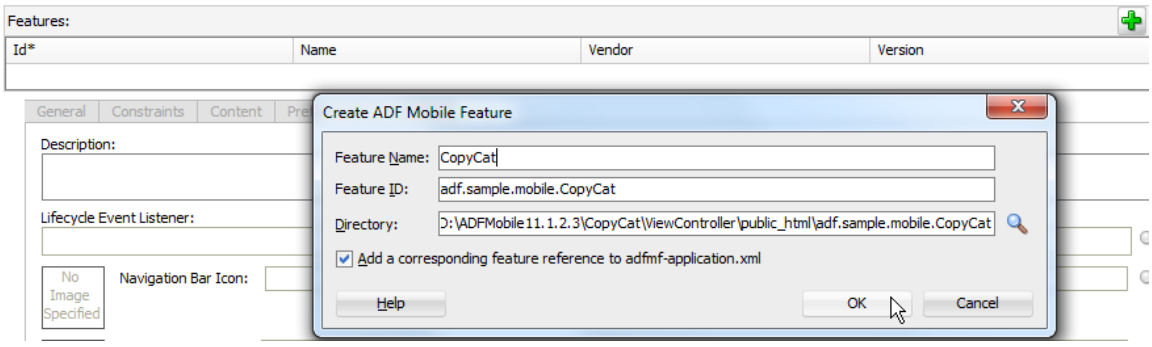


Once you finished the new application wizard, choosing the application name as **CopyCat** and the default package as **adf.sample.mobile**, the mobile application project structure is generated as shown below.

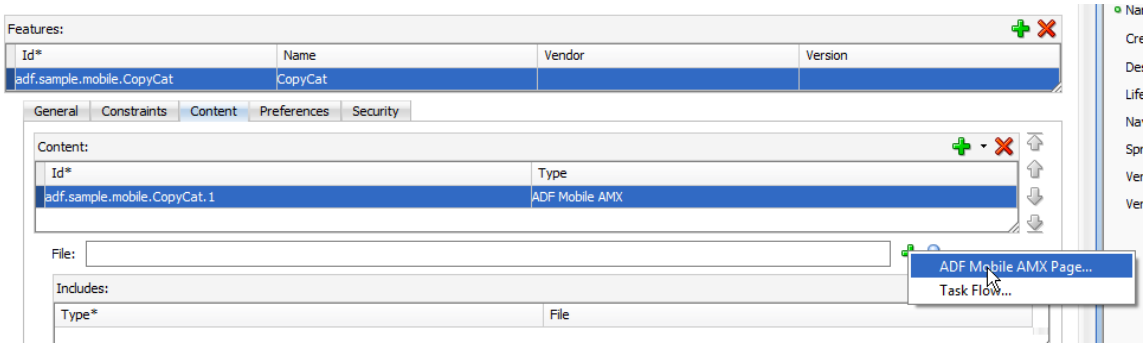


With the **adfmf-feature.xml** file of the **ViewController** project open, click the green plus icon next to the **Feature** label to create a new feature. Remember that features are reusable mobile interface units that are listed on the mobile application menu.

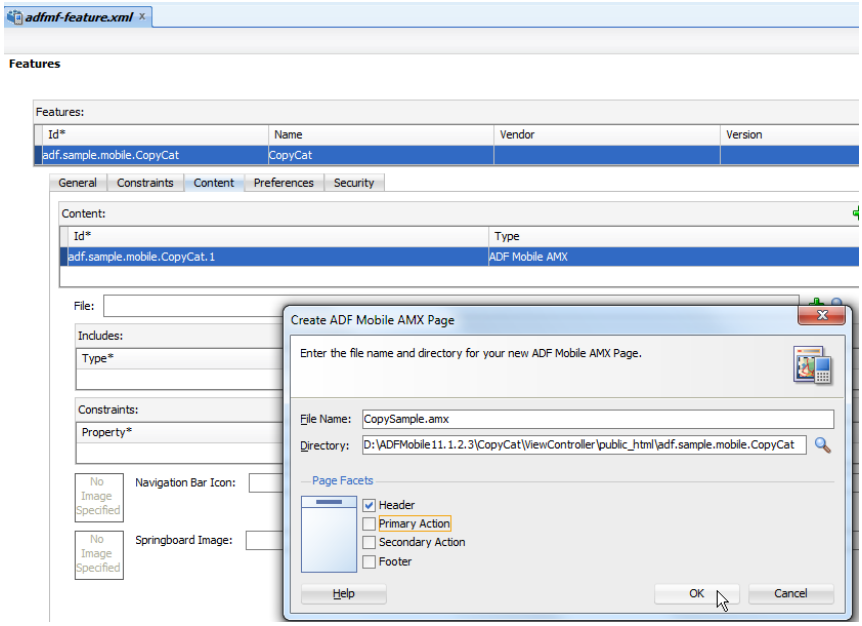
As shown below, name the feature as the application: **CopyCat**



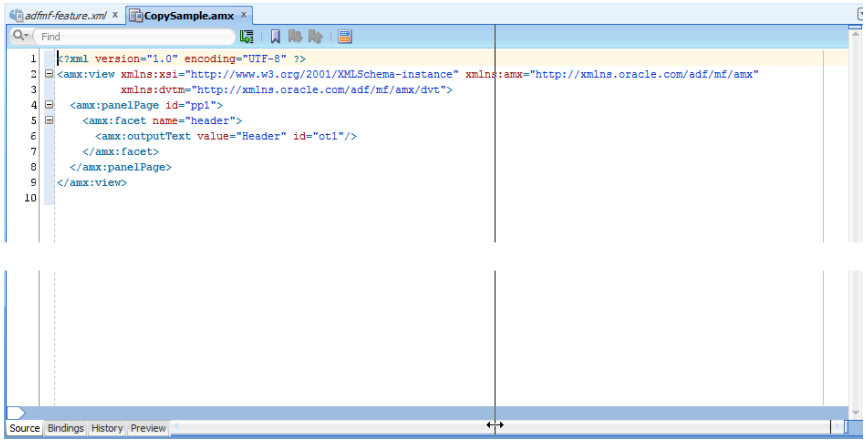
Next, click the gree plus icon next to the File field in the **Content** tab of the feature definition and choose **ADF Mobile AMX** page.



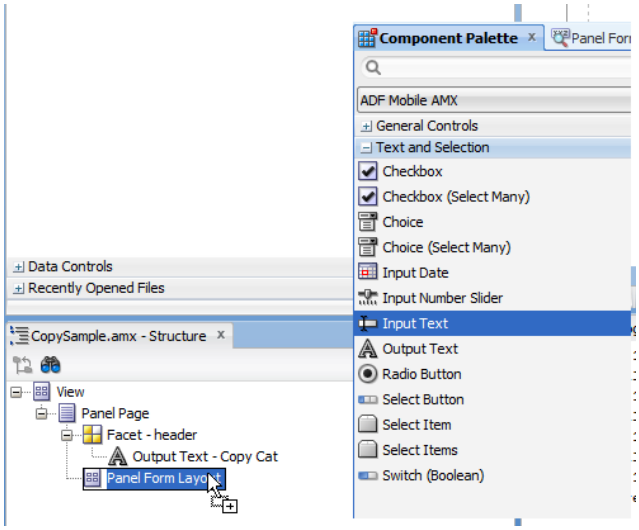
Name the AMX page **CopySampleand** de-select all page options except for the **Header** section.



The AMX page opens in the code view window. I recommend you drag the vertical window icon (see image below) to split the editor view and – for the second split window – click the **Preview** tab so you get a live preview of changes you add to the AMX page.



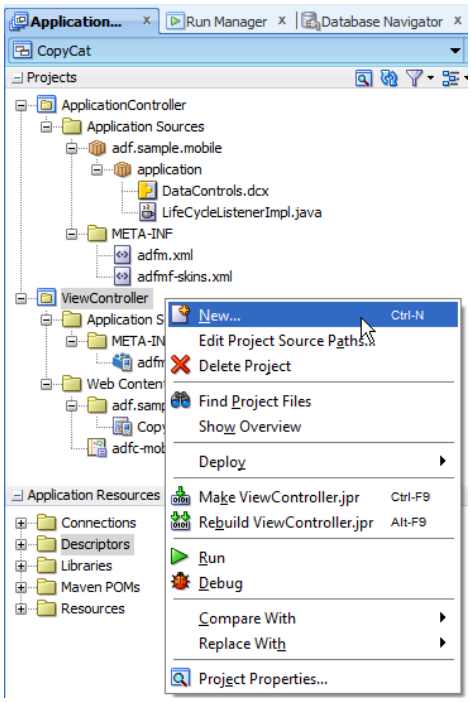
From the Component Palette drag and drop a Panel Form Layout component (Layout section) and then two Input Text components (Text and Selection section). Optionally, add a command button for a simulated model update.



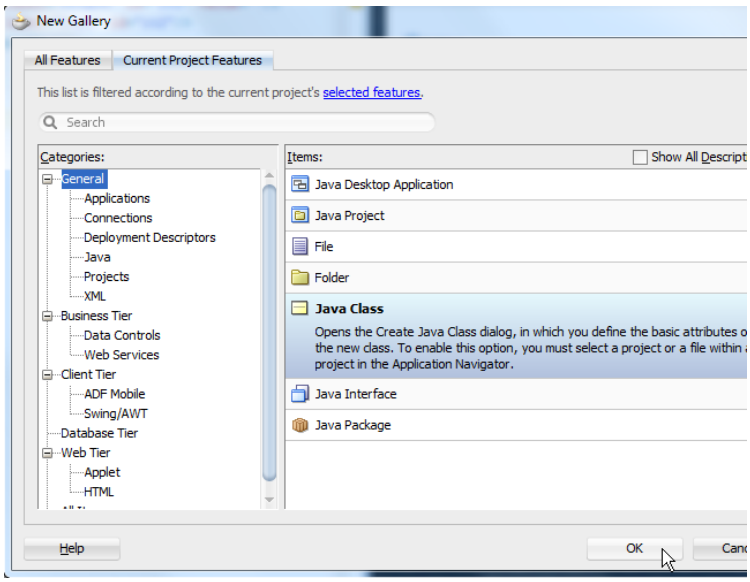
For the synchronization between the two input fields, you need to link the Input Text component **value** properties to a managed bean setter/getter pair (JavaBean property). The managed bean is what you create next.

Building the Managed Bean

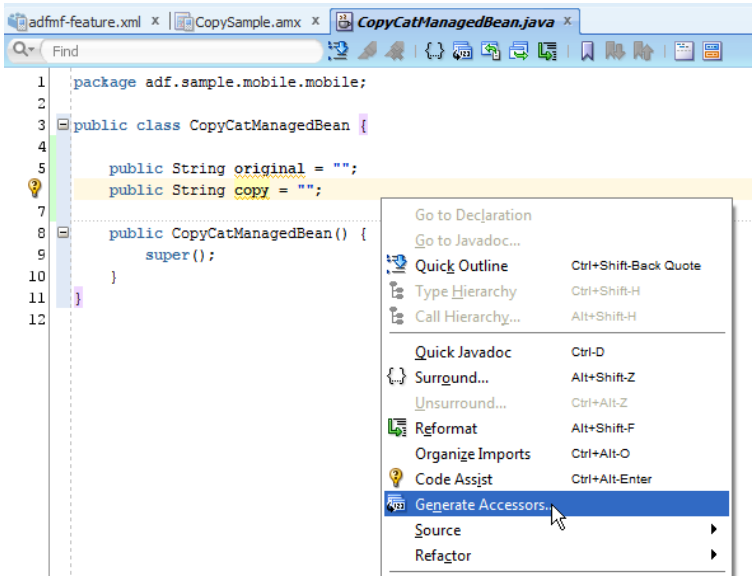
To build a managed bean, select the **ViewController** project node with the right mouse button and choose **New** from the context menu as shown in the image below.



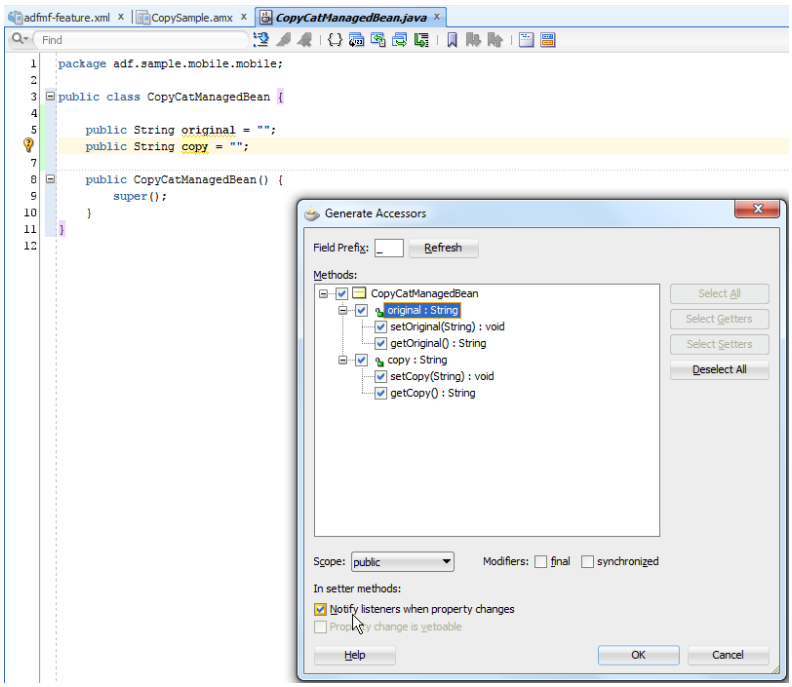
Create a new Java class by selecting the **General -> Java Class** option and pressing OK.



In the Java class editor, create two properties **original** and **copy** as shown in the image below. Then click anywhere in code editor using the right mouse button and choose **Generate Accessor** from the context menu.



In the **Generate Accessors** dialog, select the option to generate setter and getter methods for each of the properties and – **important !!!** – ensure the **Notify listeners when property changes** select box is checked. The latter ensures that a property change event is raised when the value of the properties change. ADF Mobile runtime detects this event and refreshes the UI bound to the property, which is how the synchronization feature for the UI refresh works.



The generated code is shown in the image below. You can see the **PropertyChangeSupport** code being generated, as well as the change event notification fired in each of the setter methods.

```
public class CopyCatManagedBean {

    public String original = "";
    public String copy = "";
    private transient PropertyChangeSupport propertyChangeSupport = new PropertyChangeSupport(this);

    public CopyCatManagedBean() {
        super();
    }

    public void setOriginal(String original) {
        String oldOriginal = this.original;
        this.original = original;
        propertyChangeSupport.firePropertyChange("original", oldOriginal, original);
    }

    public void addPropertyChangeListener(PropertyChangeListener l) {
        propertyChangeSupport.addPropertyChangeListener(l);
    }

    public void removePropertyChangeListener(PropertyChangeListener l) {
        propertyChangeSupport.removePropertyChangeListener(l);
    }

    public String getOriginal() {
        return original;
    }

    public void setCopy(String copy) {
        String oldCopy = this.copy;
        this.copy = copy;
        propertyChangeSupport.firePropertyChange("copy", oldCopy, copy);
    }

    public String getCopy() {
        return copy;
    }
```

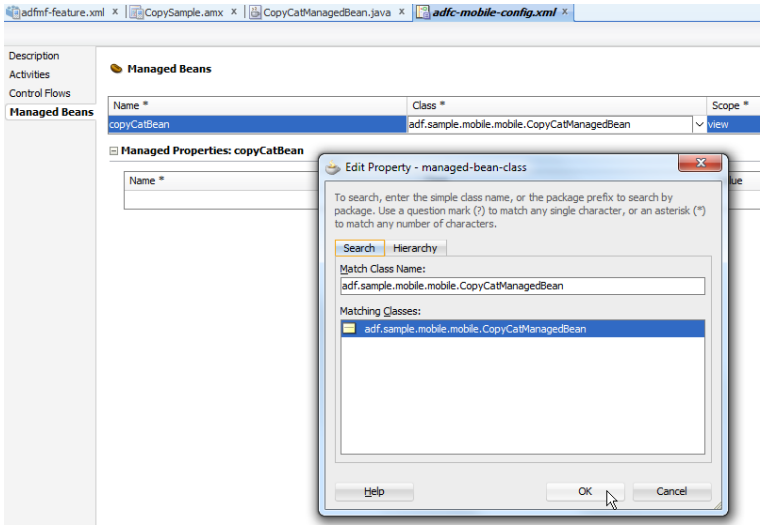
The only manual edit you have to do is to add…

this.setCopy(original);

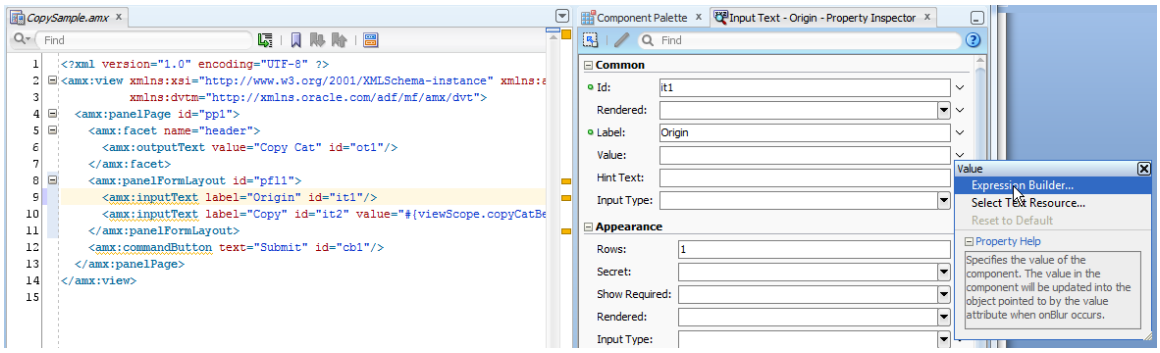
… to the **setOriginal** method. Each time the original field value is changed, the setCopy method is called, which then itself notifies registered listeners (ADF Mobile runtime in this case) about the change so a UI refresh happens.

```
public class CopyCatManagedBean {

    public String original = "";
    public String copy = "";
    private transient PropertyChangeSupport propertyChangeSupport = new PropertyChangeSupport(this);

    public CopyCatManagedBean() {
        super();
    }

    public void setOriginal(String original) {
        String oldOriginal = this.original;
        this.original = original;

        this.setCopy(original);

        propertyChangeSupport.firePropertyChange("original", oldOriginal, original);
    }
```

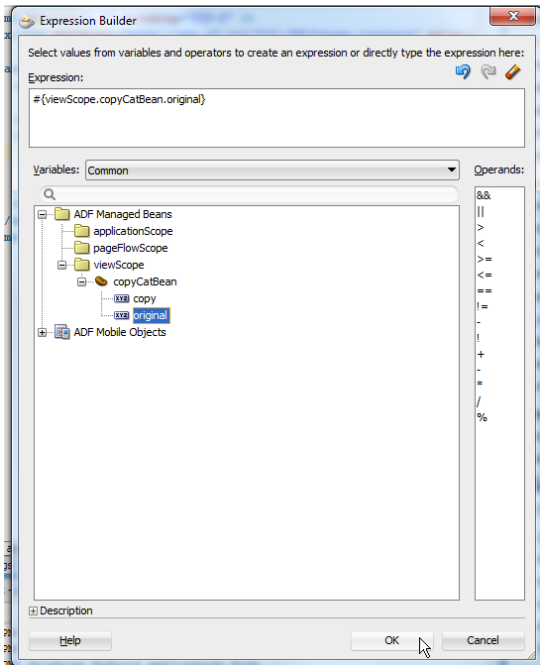
To configure the Java class as a managed bean, open the **adfc-mobile-config.xml** file in the JDeveloper application navigator (double click or right mouse click) and – in the configuration editor – switch to the **Managed Beans** category. Here, click the green plus icon (not show in the image below) to create a new managed bean configuration. The Java class built before can be looked up using the **Class** field (if you click the arrow icon to the right, you can choose the edit option to search for the class).



With the managed bean created in **viewScope** (we don't need any longer scope than that) you can select the Input Text components and use the JDeveloper Property Inspector to declaratively configure the component **value** property to point to the **origin** and **copy** property of the managed bean.

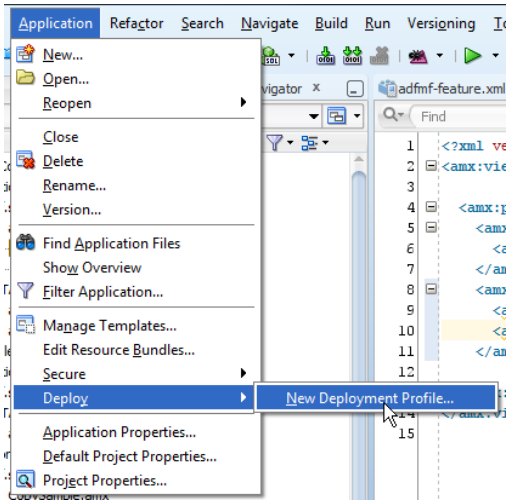


The Expression Language editor is shown in the image below. As you can see, the managed bean is listed in the **ADF Managed Bean** section under its scope (**viewScope)**

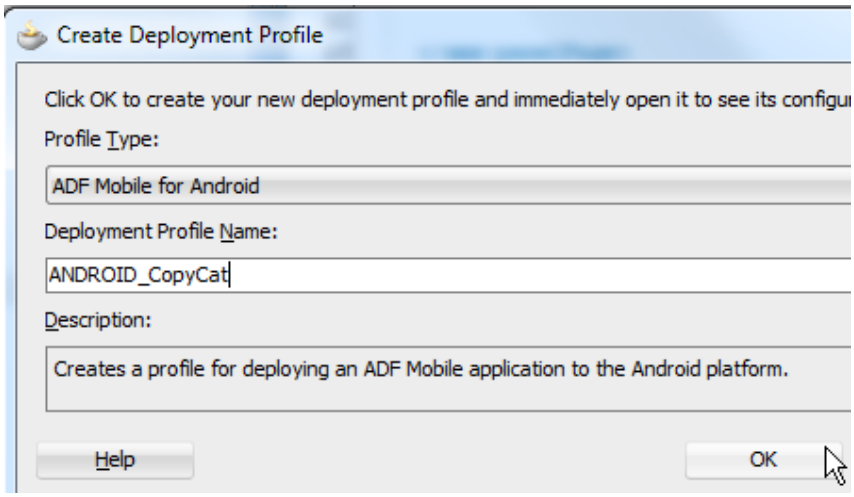


Deployment

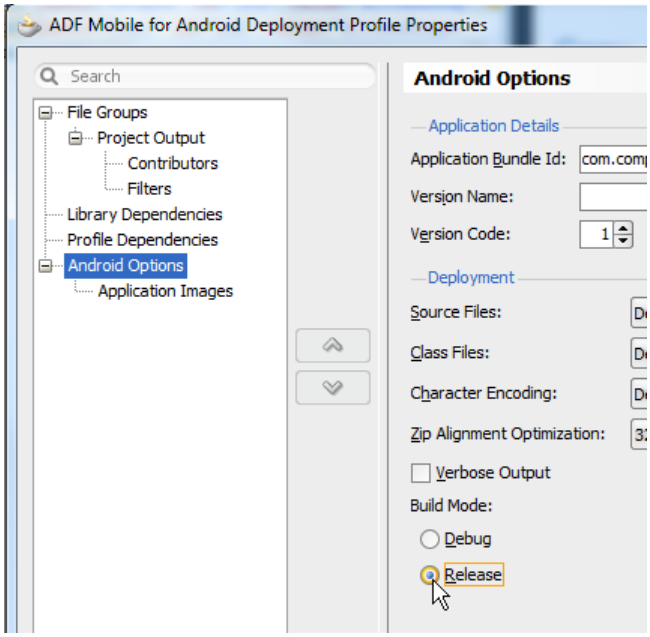
To deploy the sample application, choose **Application -> Deploy -> New Deployment Profile** from the JDeveloper menu.



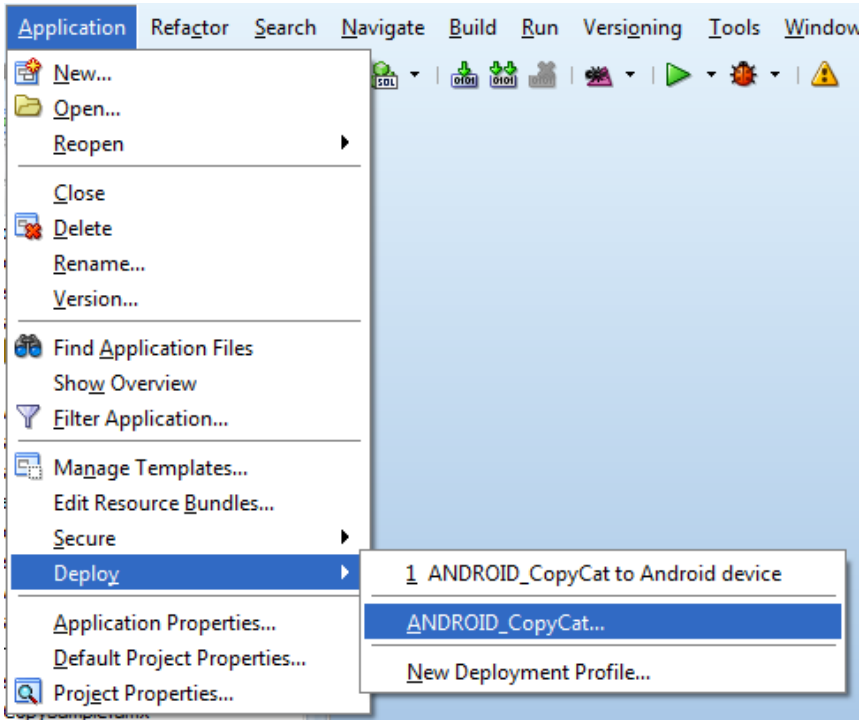
Select the target mobile platform (**ADF Mobile for Android**) from the **Profile Type** select list and provide a meaningful name for the deployment.



In the **Profile Properties** dialog, expand the **Android options** (if deploying to Android)and select the **Release** option, which provides better performance at runtime.



Finally, with the mobile device connected to your laptop, choose **Application -> Deploy -> <Profile Name>** from the JDeveloper menu to deploy the application. This may take 1-2 minutes the first time you deploy the application.



Android

ADF Mobile allows mobile on-device applications to be developed for Apple iOS and the Android platform (plus what will come in the future). From a development perspective this means that a single code line does it. So choosing between the two mobile platforms I decided for Android as the platform that I use for ADF Code Corner mobile sample development and testing. The development platform however should not matter as ADF Mobile is doing the cross-platform compilation trick for you.

However, from a perspective of what has been tested and where have samples been tested on, the answer is Android 2.3 and Android 4.0 for mobile phone and tablet.

RELATED DOCOMENTATION

☒	
☒	
☒	