ADF Mobile Code Corner

m09. How-to work with user preferences

twitter.com/adfcodecorner

Abstract:

User preferences are settings that the mobile phone tracks beyond application re-starts. You use them in your mobile application to allow users to "configure" a mobile application to their needs.

This article shows how to define user preferences and how to access preference setting from within the ADF Mobile application.

Author: Frank Nimphius, Oracle Corporation
twitter.com/fnimphiu
10-MAY-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

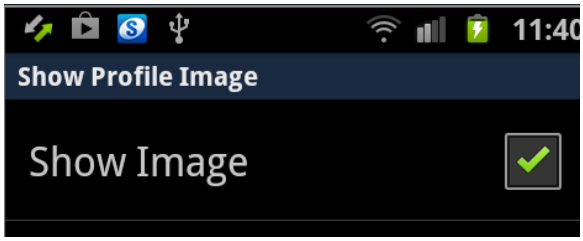
Introduction

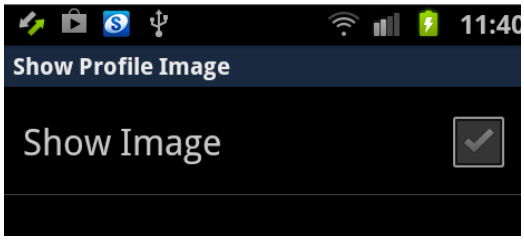
Preferences are handy when working with mobile applications. Imaging a mobile application that queries a rmote yellow book service and thus may come with a lareger result set. Based on the quality of the Internet access you have, and for better performance, you want to exclude images from the query result. Though not as sophisticated, the sample you download for this article, and that I explain how to build in the following, alowsa you to switchthe display of images in a profile form on/off.

The image above shows a mocked-up profile page that shows a large image associated with the user profile. Though I personally love this image, in the following I explain how you can use ADF Mobile application preferences to switch this image on and off.

In the sample (shown in the image above for an Android device) you use the native application menu to show the **Preferences** screen.

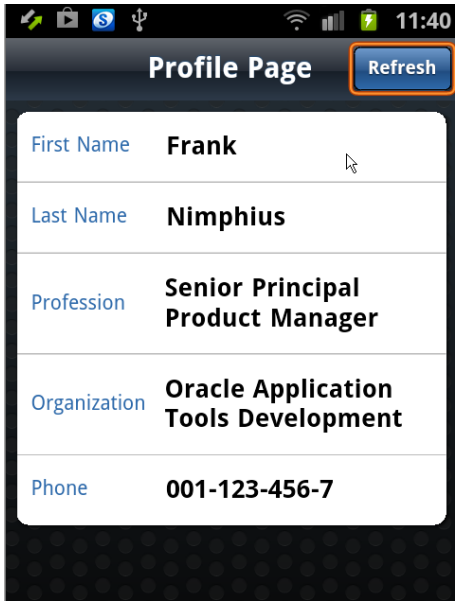
The preference screen is shown in the image below and renders in the mobile device native look and feel (so it will look different on iOS)





In this sample, after returning from the preferences screen, you hit the **Refresh** button to switch the image **on** or **off** according to the preference setting. In a real application, this refresh could be invoked by a feature lifecycle listener that indicates the resume of a mobile application screen (will cover life cylce listeners in a later article).

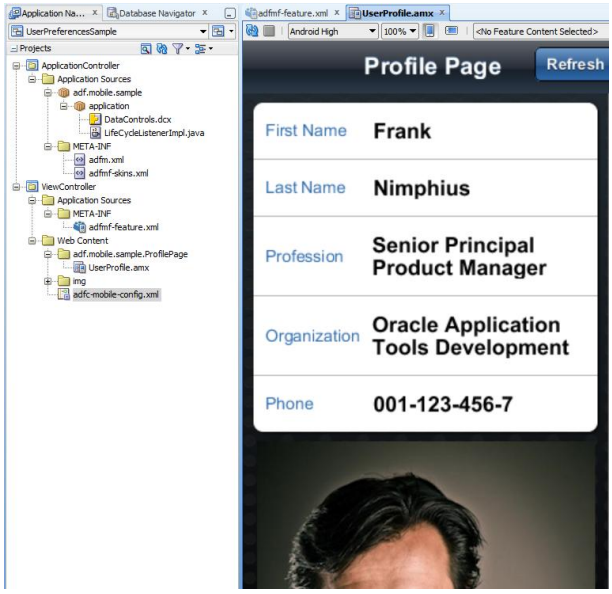
The effect of the preference change is shown in the image below.



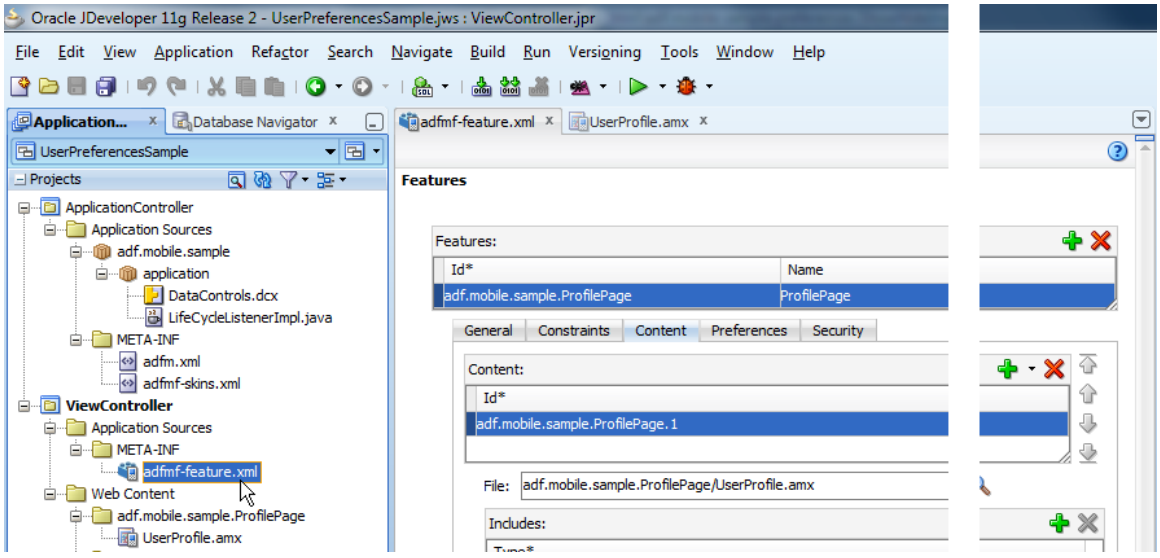
User Preference Sample

As said, the sample is a mockup of a user profile page and as such has its information hard coded (how-to query data is not part of this article and is explained in a previous ADF Mobile Code Corner article).

As you can see in the image below, the picture is enabled by default.

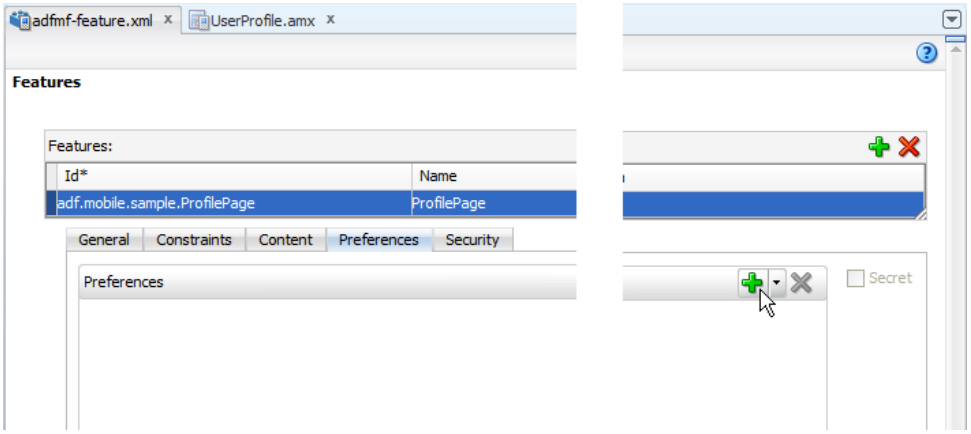


Preferences in ADF Mobile can be defined for the application as a whole and for individual features. The latter makes sense if you plan to reuse your mobile functionality (which is what "Features" are for). In this sample I used preferences on the feature level because profile pages sound like a use case worth to reuse (though not as a mockup ;-))

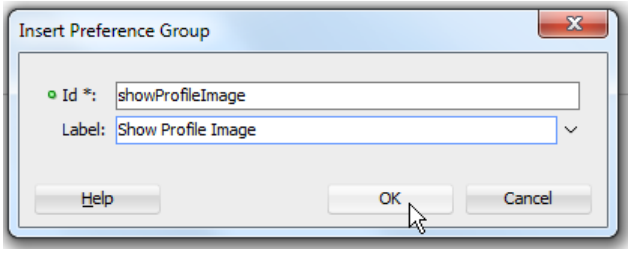


For features, the preference is configured in the adfmf-feature.xml file for which you see the visual editor shown in the right part of the image above.

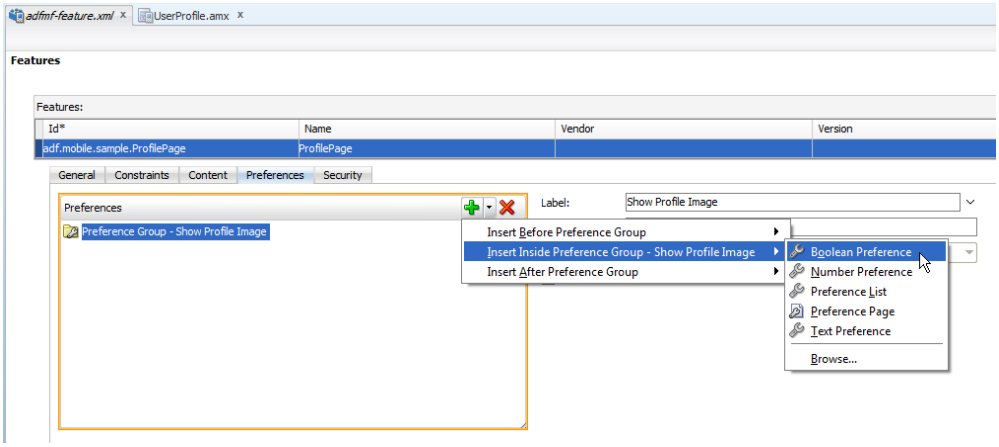
You create a new feature by pressing the green plus icon next to the **Preferences** label.



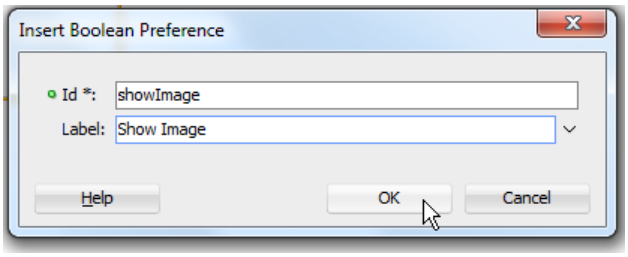
The ID of the feature should be chosen unique, whereas the only requirement for the display label is to be short and descriptive



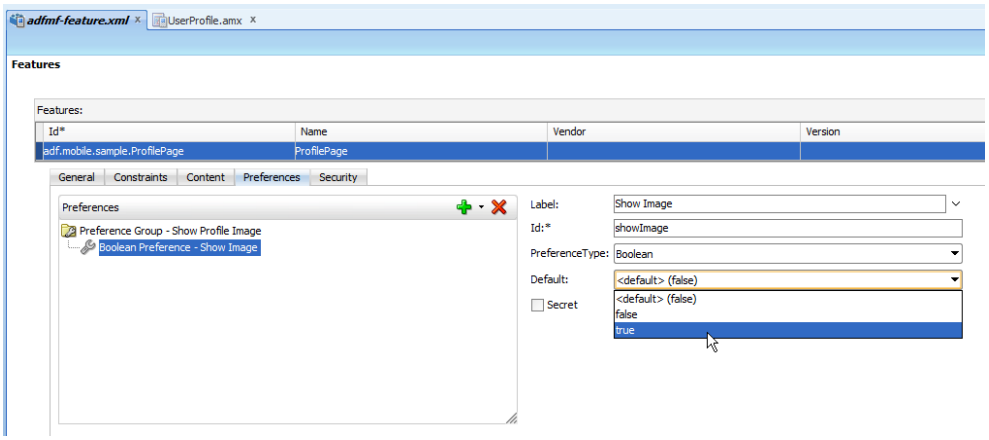
Once the preference is created, you select it in the **Preferences** field and press the green plus icon as shown in the image below.



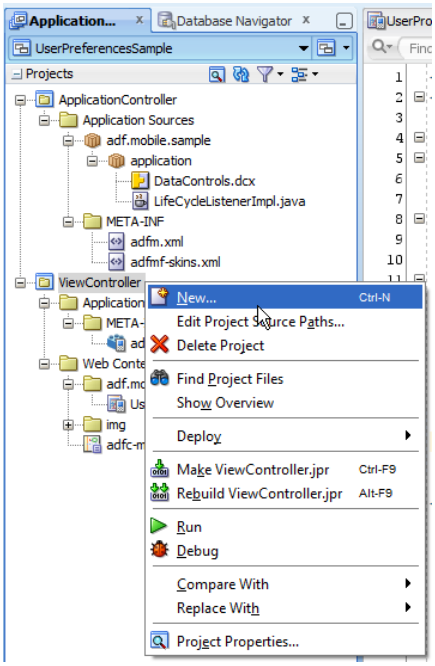
You can create boolean, numeric and free text preferences. In addition, choosing the **Preference Page** entry you can create category preferences (e.g. for the user profile use case you can have a category called **information display** (Preference Page) that lists all the available profile information item (e.g. first name, last name, profession, organization etc.) for the user to "tune" the display of user information, which can be a benefit if your screen size is small (even for mobile phones, the hardware vendors stepped back from the bigger and larger mantra to what they call "mini" – which in fact the retro size of phones as we had them in the past). So preferences help users to optimize screen displays where adaptive and responsive design falls short.



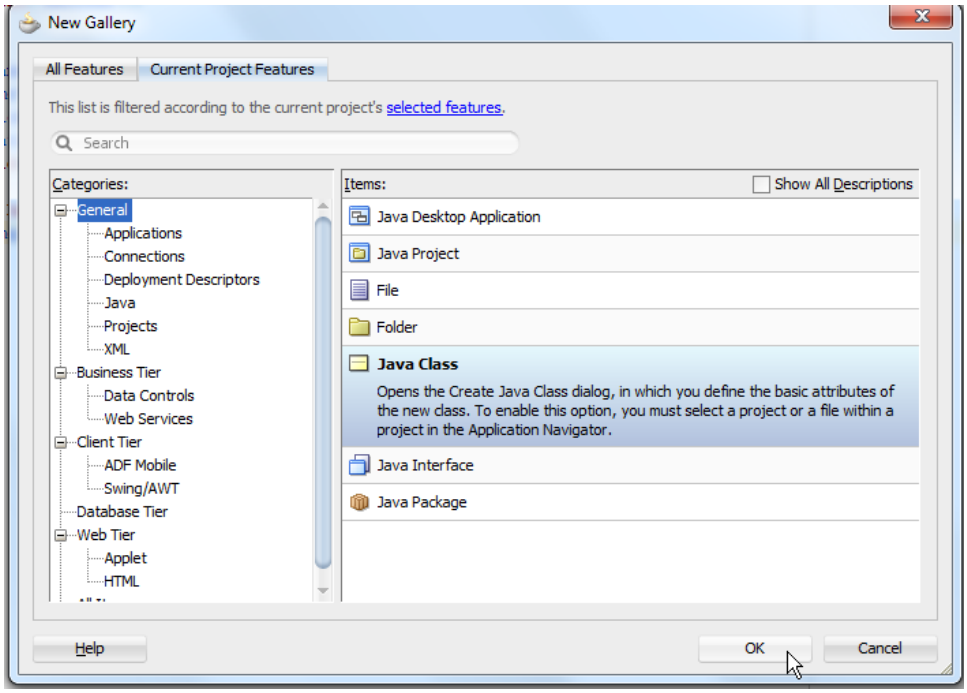
In the example, I am using a boolean preference to switch the visibility of the image on and off.



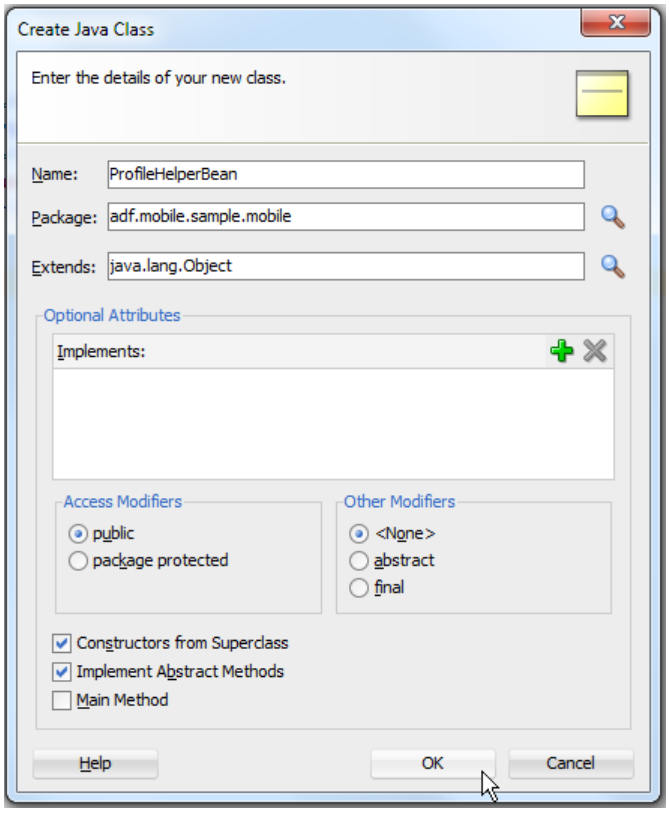
Also declarative (shown in the image above) you set the default value of the preference item you created. Later, the actual preference value can be accessed from Expression Language and Java. In this sample I use Java because it allows me to use the Java Bean `PropertyChange` notification to refresh the screen so the preference settings are displayed.



In the next step thus I create a new managed bean that exposes a property for the component **inlineStyle** property that determines showing or hiding of the image. In a use case where you need to suppress the query of images (a functional constraint), you would access the preference in the logic that defines the query.

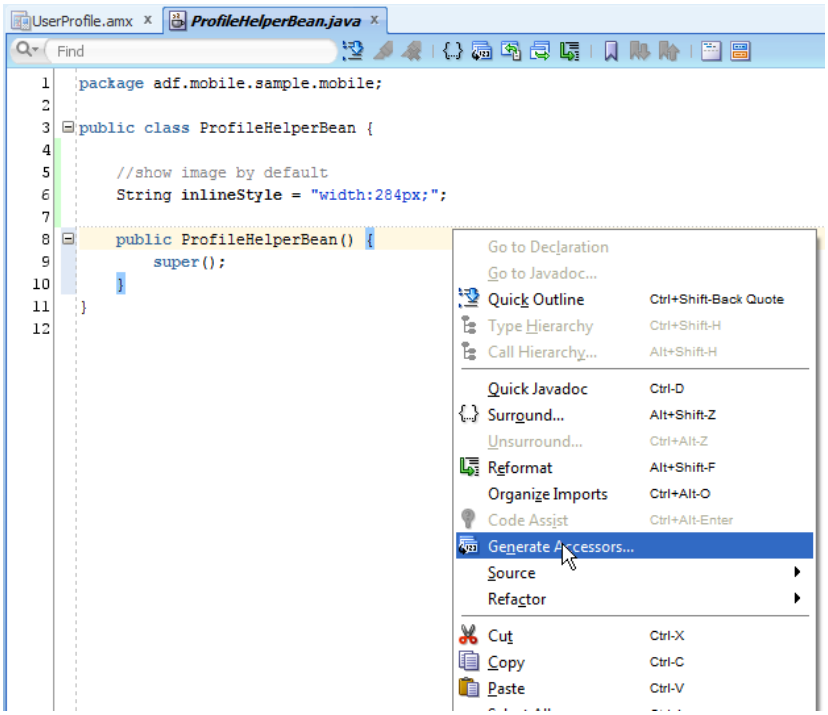


The image below shows the Java class dialog with the Java bean definition.

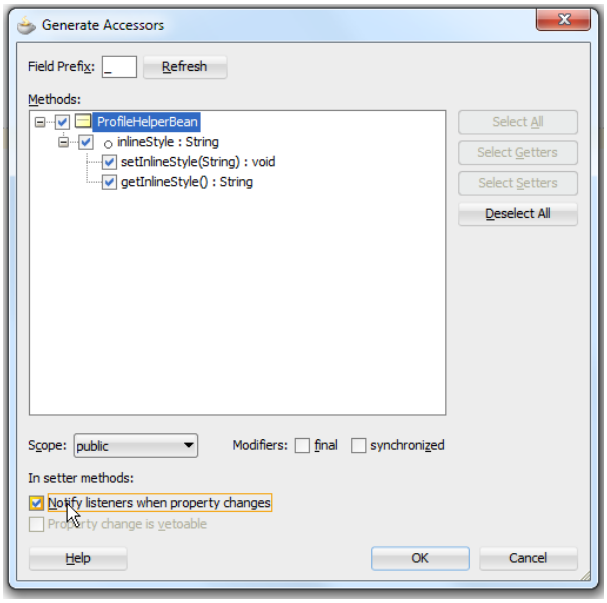


One property, **inlineStyle**, is created that returns the CSS for the item to show/hide

Note: If you use the same CSS for many items, I recommend using skinning in ADF Mobile together with the **styleClass** property on the components.

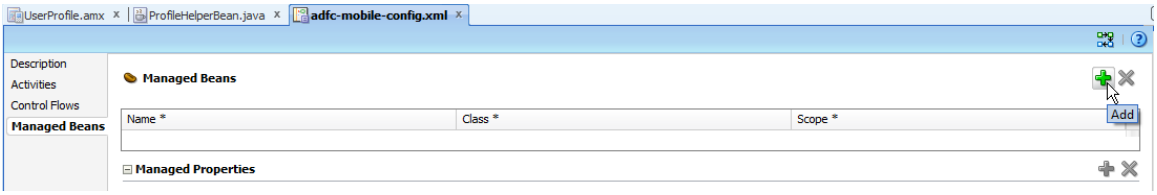


Generate accessor (setter / getter method) so the property becomes EL accessible.

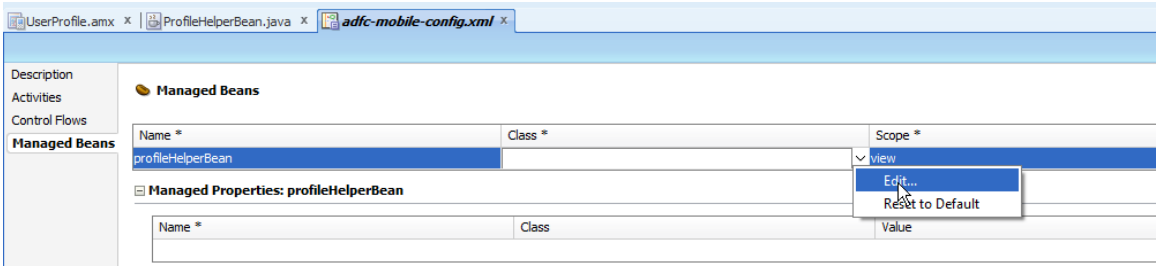


Important when working with Java properties in ADF Mobile, ensure the **"Notify listeners when property changes"** option is selected. Enabling this change notification will tell the ADF Mobile framework when to refresh a component on the screen.

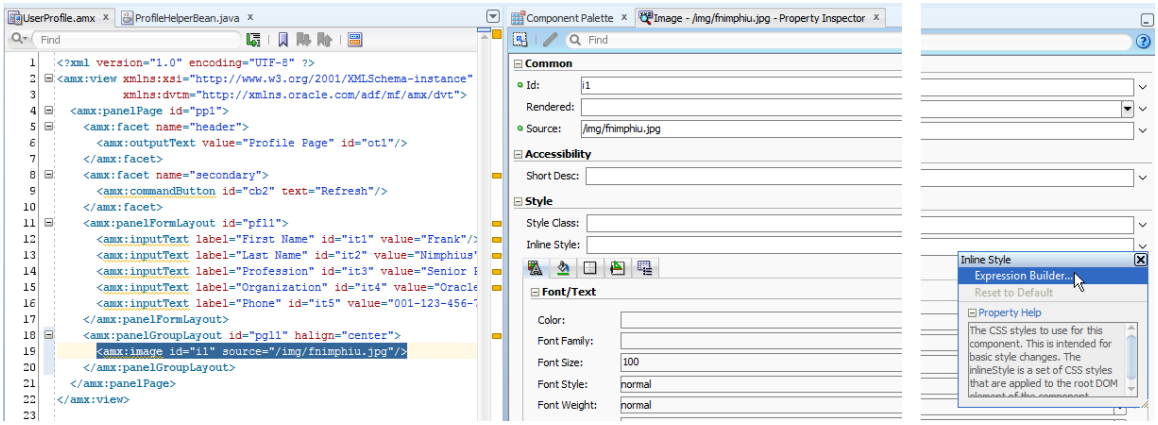
Next you configure the Java class as a managed bean in the **adfc-mobile-config.xml** file, or if your feature is based on a bounded task flow, the bounded task flow configuration file. The sample in this article only uses a single page and thus the **adfc-mobile-config.xml** is used.



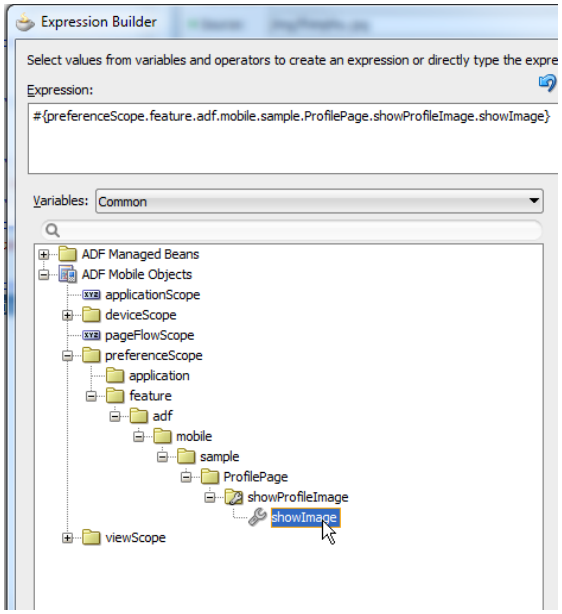
To configure a managed bean you define a name (which then appears as a source in the EL editor), lookup the Java class and define a scope (which for this use case is **view**)



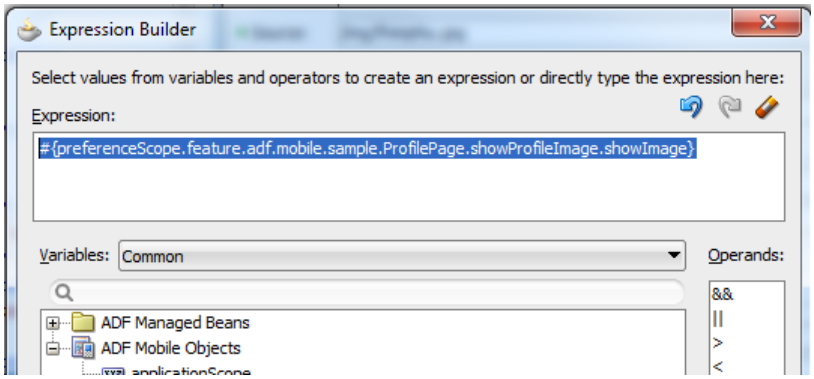
Next you need to build the EL expression to access the preference. This sample needs the information in Java, so using the **inlineStyle** property (as shown in the image below) is just a simple and declarative way to compose the expression ;-)

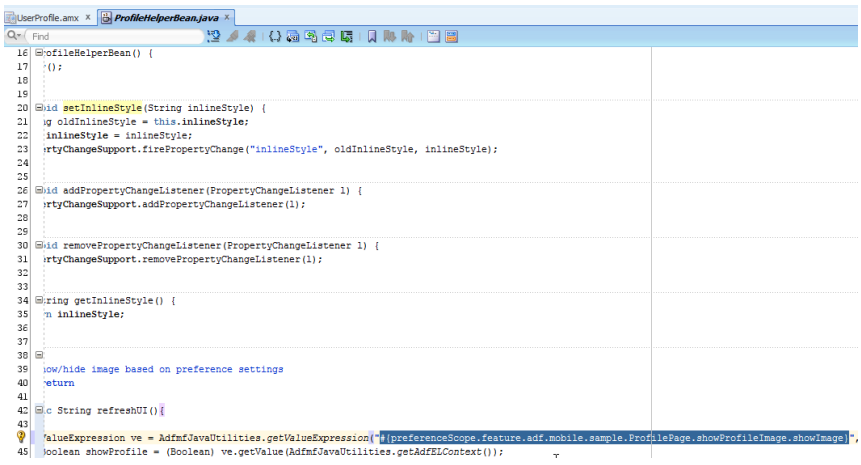


As you can see in the image below, the **preferenceScope** scope is displayed as a node under the **ADF Mobile Object** node. Preferences can be created on the application level, where they are defined in the adfmf-application.xml file, and the feature level. The path to a preference defined in a feature includes the feature ID and the preference ID. To get the preference EL expression, select the preference as shown in the image below …



… and copy the EL to the clipboard. We will see the string later in the Java code displayed in the image at the end of this page.



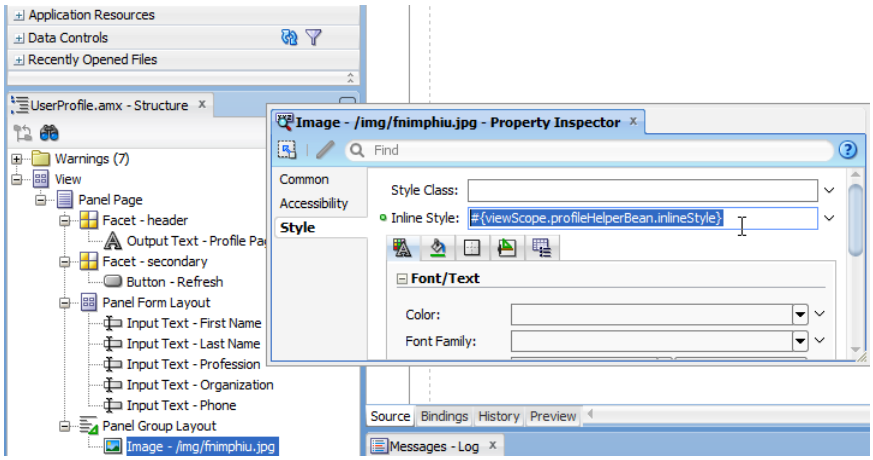


Comple JavaBean code

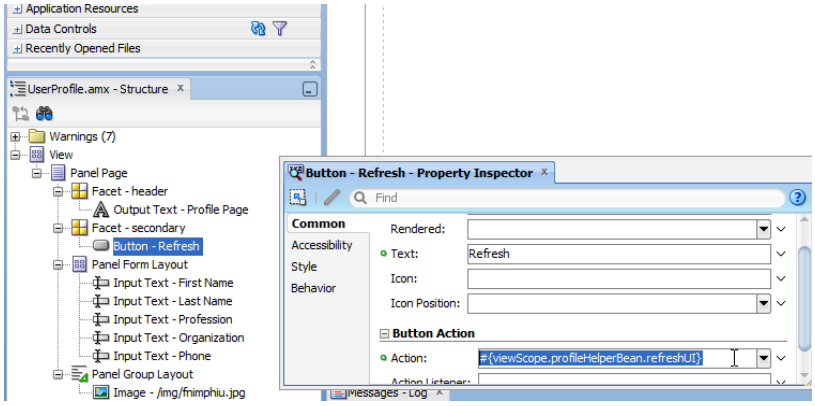
```
public class ProfileHelperBean {
 //show image by default
 String inlineStyle = "width:284px;";

 private transient PropertyChangeSupport propertyChangeSupport
          = new PropertyChangeSupport(this);
public ProfileHelperBean() {
    super();
}

public void setInlineStyle(String inlineStyle) {
  String oldInlineStyle = this.inlineStyle;
  this.inlineStyle = inlineStyle;
  propertyChangeSupport.firePropertyChange(
            "inlineStyle", oldInlineStyle, inlineStyle);
}

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

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

public String getInlineStyle() {
   return inlineStyle;
}

/**
 * show/hide image based on preference settings
 * @return
 */
 public String refreshUI(){
    ValueExpression ve =
         AdfmfJavaUtilities.getValueExpression("
```

```
            #{preferenceScope.feature.adf.mobile.sample.
            ProfilePage.showProfileImage.showImage}", Object.class);

            Boolean showProfile =
              (Boolean) ve.getValue(AdfmfJavaUtilities.getAdfELContext());
            String cssStyle = "";
            if(showProfile == Boolean.TRUE){
               cssStyle ="width:284px;";
            }
            else{
               cssStyle="display:none;";
            }
            //enforce image item refresh
             this.setInlineStyle(cssStyle);
               return null;

            }
}
```

The **inlineStyle** property of the **profileHelperBean** managed bean is referenced from the **amx:image** component's inlineStyle property as shown in the image below.



The **refresh** command button references the **refreshUI** method and sets the CSS for the inline style property. The UI component refreshes in response to this and either hides or shows the image.

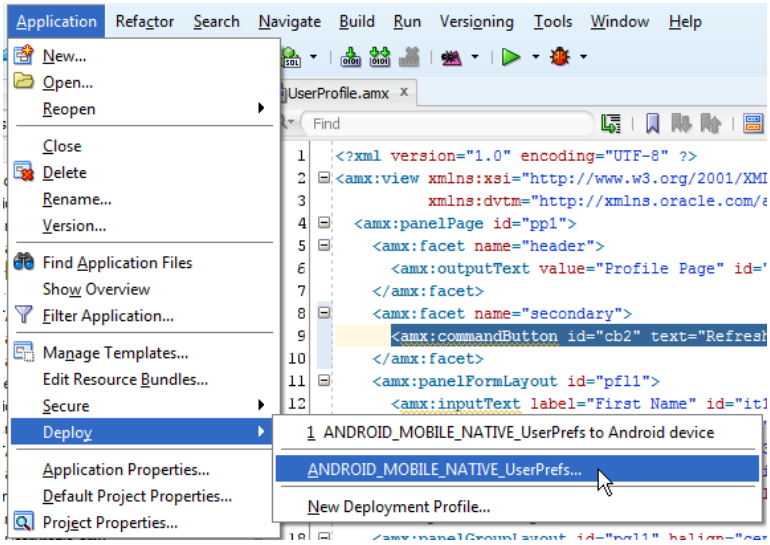


Page Source

```
<amx:panelPage id="pp1">
  <amx:facet name="header">
    <amx:outputText value="Profile Page" id="ot1"/>
  </amx:facet>
  <amx:facet name="secondary">
    <amx:commandButton id="cb2" text="Refresh"
                                    action="#{viewScope.profileHelperBean.refreshUI}"/>
  </amx:facet>
  <amx:panelFormLayout id="pfl1">
    …
  </amx:panelFormLayout>
  <amx:panelGroupLayout id="pgl1" halign="center">
    <amx:image id="i1" source="/img/fnimphiu.jpg"
                          inlineStyle="#{viewScope.profileHelperBean.inlineStyle}"/>
  </amx:panelGroupLayout>
</amx:panelPage>
</amx:view>
```

Deployment

To deploy the sample, choose the **Application -> deploy** menu in Oracle JDeveloper and follow the ANDROID_MOBILE_NATIVE … item. If you want to deploy to iOS, follow the **New Deployment Profile** entry to create an iOS deployment profile.



Sample Download

You can download the JDeveloper 11.1.2.3 workspace of this sample as sample **"m09"** from the ADF Code Corner website. No special setup is required, except for configuring your JDeveloper IDE for mobile deployment, which you find documentation for in the list of related documentation.

http://www.oracle.com/technetwork/developer-tools/adf/learnmore/index-101235.html#CodeCornerMobileSamples

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. Also note that – at least on my Android mini tablet – the preference window did not show properly, which may be a problem with my device.

It worked on the phone though.

RELATED DOCOMENTATION

☒	ADF Mobile Set-up Tutorial http://docs.oracle.com/cd/E18941_01/tutorials/MobileTutorial/jdtut_11r2_54_1.html
☒	ADF Mobile Tutorial http://docs.oracle.com/cd/E18941_01/tutorials/BuildingMobileApps/ADFMobileTutorial_1.html
☒	User preferences http://docs.oracle.com/cd/E35521_01/doc.111230/e24475/userpreferences.htm#CJAGHGCG