ADF Mobile Code Corner

m08. How-to access values of the selected list rows in a SelectManyChoice component

twitter.com/adfcodecorner

Abstract:

In select many choices based on Oracle ADF bindings, the returned values are the indexes of the selected values, not the values itself. In ADF we do this so developers can also access objects values from a list, which would not be possible if the value itself needs to be provided with the rendered list component. A common ADF requirement, not only for ADF Mobile, thus is how to read the real values for a selection. This article shows how, in ADF Mobile, you access the selected row value of a select many choice list component.

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

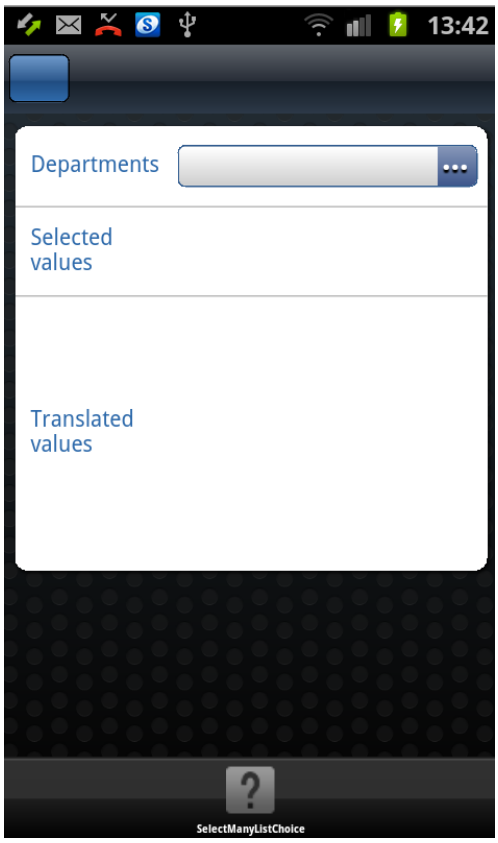
Oracle ADF Mobile Code Corner is a blog-style series of how-to documents targeting at Oracle ADF Mobile that provide solutions to real world coding problems. ADF Mobile Code Corner is an extended offering to ADF Code Corner

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.

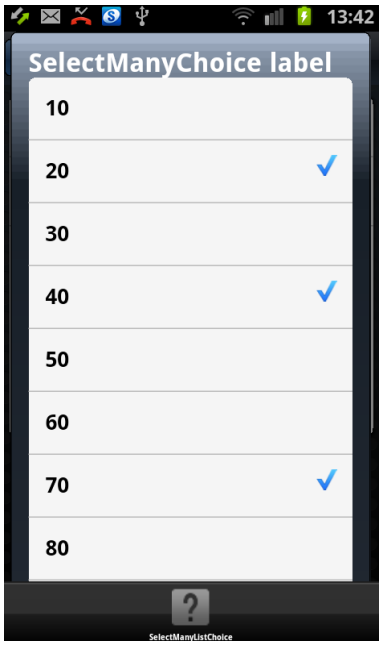
Please post questions or report problems related to the samples in this series on the OTN forum for Oracle JDeveloper: http://forums.oracle.com/forums/forum.jspa?forumID=83

Introduction

The image below shows an ADF view with a select many choice component and two fields that show the index of the selected values and a value read from the selected value row. The sample reads the from the Oracle HR schema's departments table through a WS. The **translated values** are the department names of the selcted list rows. However, as mentioned before, any attribute of the list row object can be read inresponse to a user interaction with the list.



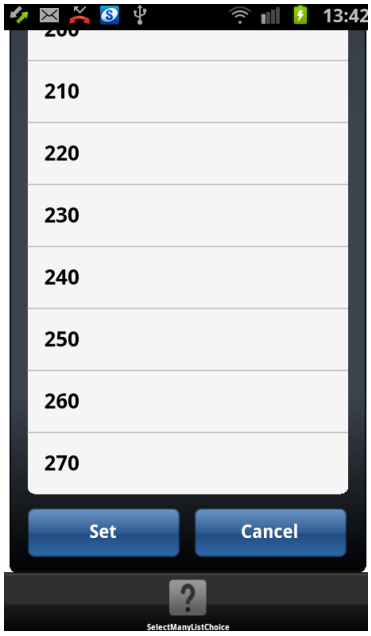
In the image below, the departments with the ID **20, 40,** and **70** are selected by a user.



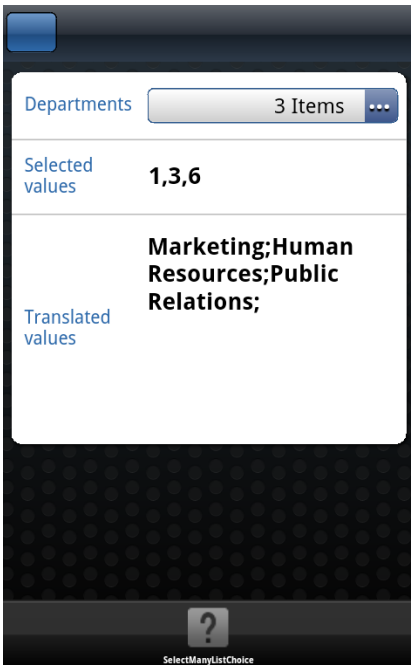
Next, the user scrolls down to the bottom of the list to confirm the value selection or to cancel it.

Note & Disclaimer: This sample has a long list of values, which in a serious production system I would have reduced by a query filter. I don't think that using a select list for 100+ rows is best pactices on a mobile device. However, this is a sample and the focus matters, which is on accessing the list selections real data values.

So when you download and run the sample, make sure you press the **Select** button.



As a result, the selected index values (starting from 0) are displayed in the **Selected values** field, as well as the department names in the **Translated values** field.



The remaining of this article explains how you "translate" seleced list values ot the real value, where you can download the sample project from and how to configure and run it.

Configuring the Web Service for this Sample

The sample for this article queries data from the Oracle HR schema (though I could have used locally mocked-up data, ADF Code Corner samples prefer data queried from the database HR schema).

The HR schema is accessed by an EJB model that is exposed as a SOAP Web Service. The Web Service project is provided as a separate JDeveloper 11.1.2.3 workspace so it can be deployed to the integrated WLS server.

After downloading the sample **m08** ZIP file from ADF Code Corner …

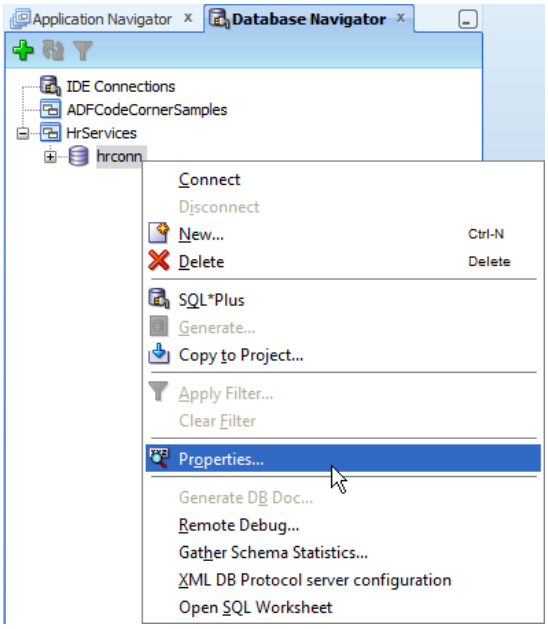
http://www.oracle.com/technetwork/developer-tools/adf/learnmore/index-101235.html#CodeCornerMobileSamples

… continue as explained below.

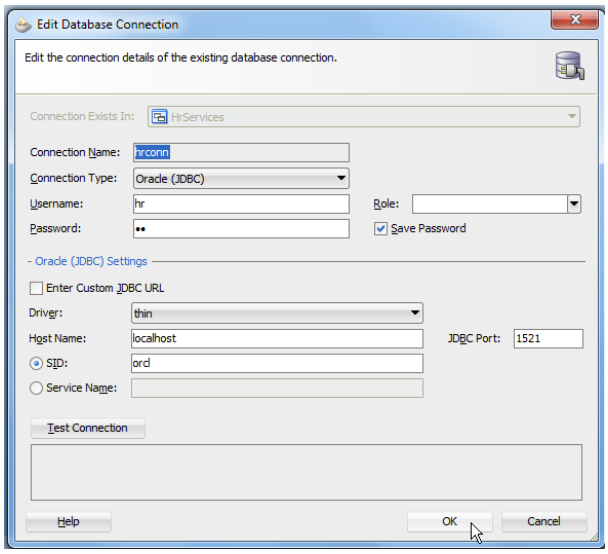
Open the Web Service workspace in JDeveloper by choosing **File -> Open** in Oracle JDeveloper 11.1.2.3 and navigate to the directory in which you unzipped the downloaded sample. Select the **HrServices.jws** file in **ADFCodeCornerSamples -> HrService** folder and press **Open**.

To configure the database access for the Web Service you need to switch to the **Database Navigator** view. For this, in JDeveloper, select the **View -> Database -> Database Navigator** menu option. Then

expand the **HrService** entry and select the contained **hrconn** configuration entry using the right mouse button.

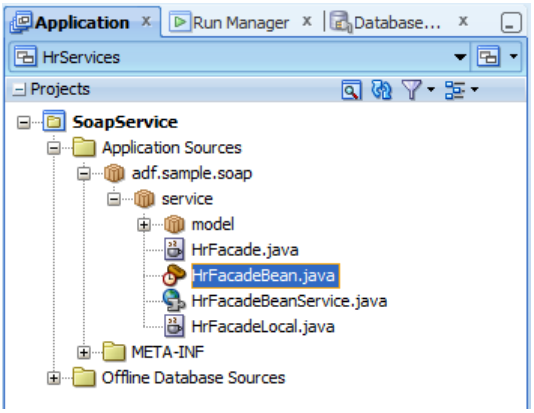


Choose **Properties** from the opened context menu to edit the database connect information as shown in the image below. Keep the name of the connection as **hrconn**.

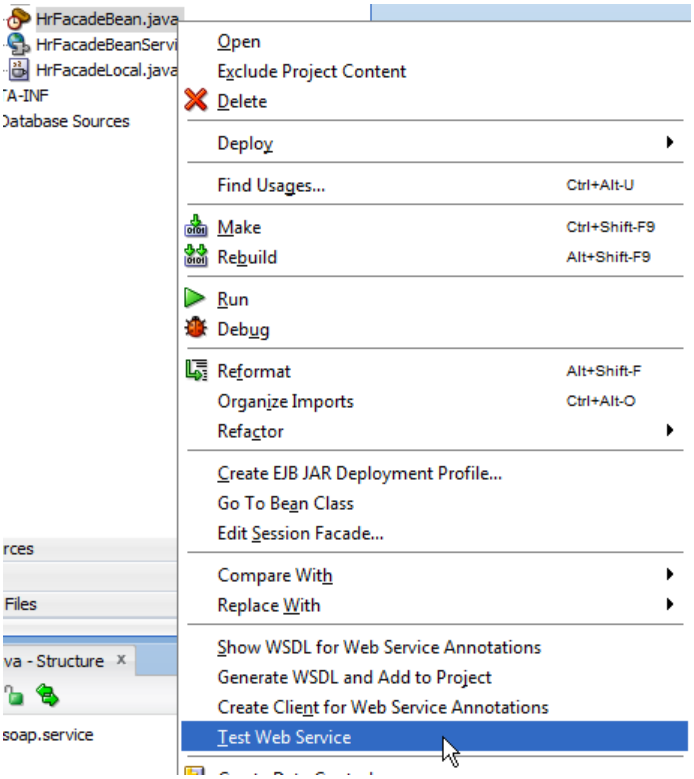


Use the **Test Connection** button to test the database connection. If the connection works, continue and run the Web Service in the JDeveloper WS tester.

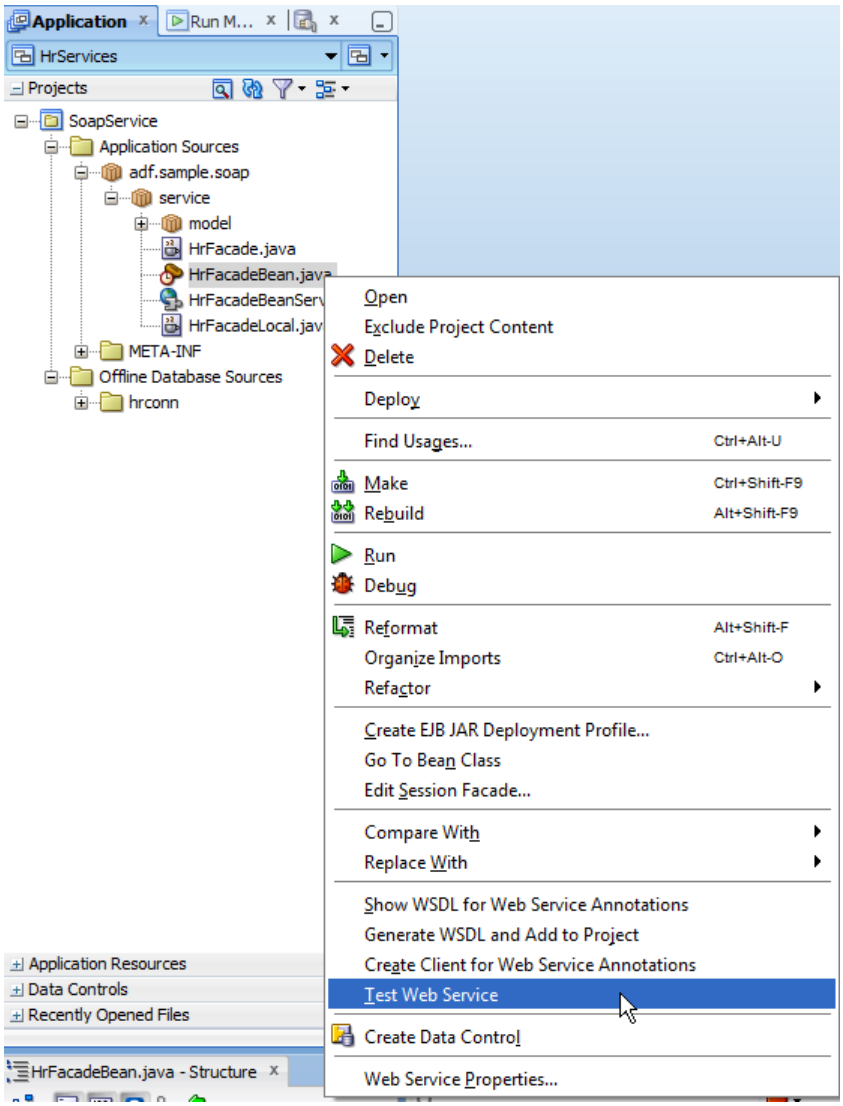
For this, in the **Application Navigator**, expand the project structure as shown in the image below and right mouse click onto the **HrFacadeBean.java** file entry.



Choose **Test Web Service** from the context menu like shown in the image below. Dong so, you deploy the Web Service to the integrated WLS in Oracle JDeveloper and start the integrated Web Service tester for testing the service.



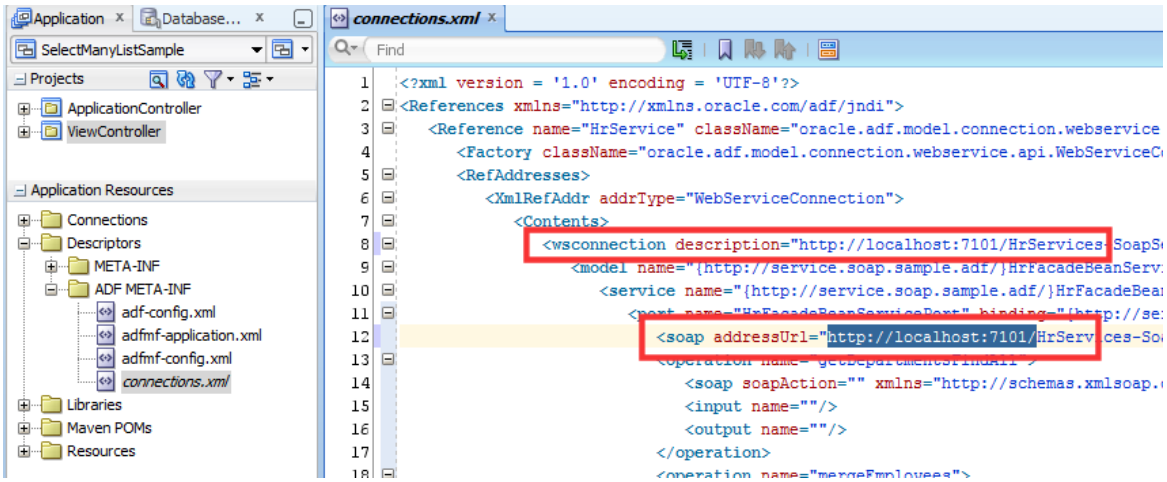
Note: If this is the first time you start the integrated WLS, a dialog shows for you to define a password for the integrated WLS administrator. Provide the password and keep all settings as defaulted. This way the integrated WLS also listens for the IP address or the domain address of your computer, which needs to be the case for the mobile device to be able to access the service.



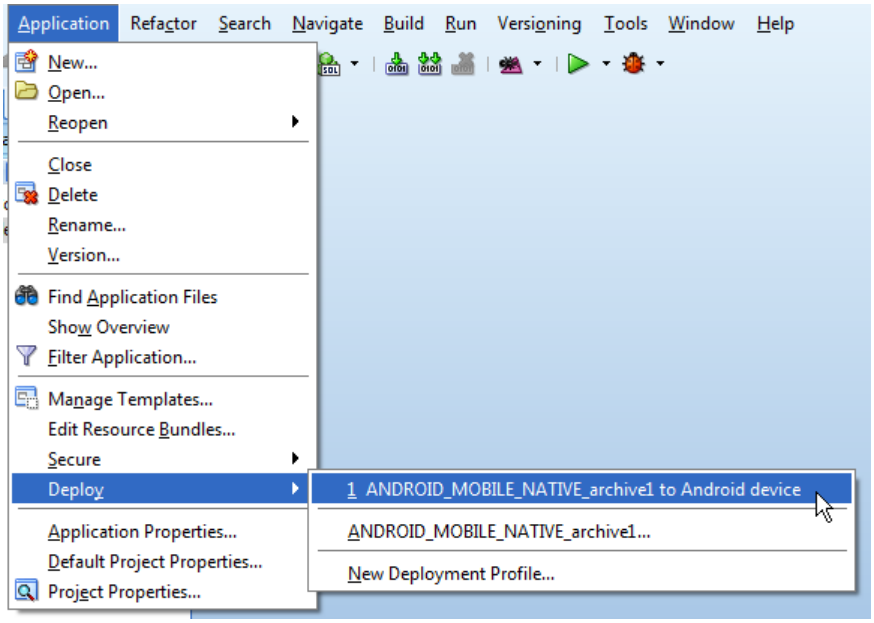
ADF Mobile Configuration & Deployment

The mobile application uses a Web Service data control to access the HR service on the server. The connect information is saved in the connections.xml file that you find in the **Application Navigator -> Application Resources -> Descriptors -> ADF META-INF** folder in JDeveloper.

Double click the **connections.xml** file to open it in the JDeveloper source editor. Change the **localhost** references in this file (marked in the image below) to the IP address or the domain name of the machine that runs the HR service. If you use the integrated WLS server in Oracle JDeveloper then you can keep the port to 7101, if not change it as well.



With this change (and the previous change of the database connection in the HRService WS project) the application is now ready for device deployment. As shown in the image below, choose the **Application -> Deploy -> ANDROID_MOBILE_NATIVE_ …** deployment profile to deploy the application to an Android device. If you want to deploy it to an Apple device, create a new deployment profile using the **New Deployment Profile** option in the same menu.



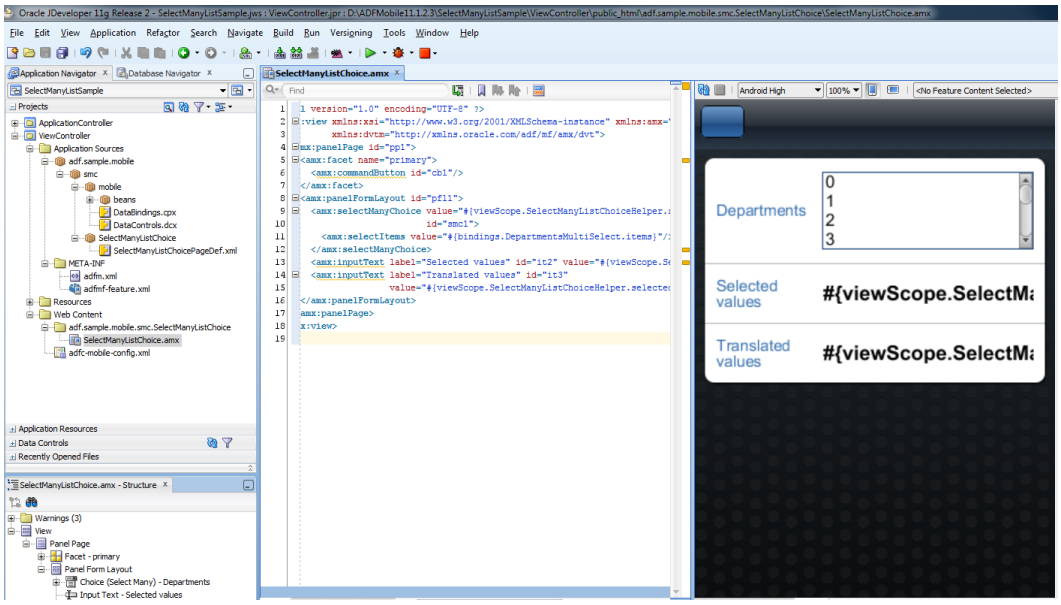
What could go wrong? If you followed the setup instructions then all that could go wrong is: i) You did not download the ADF Mobile extension and because of this the deployment fails. If so, choose **Help -> Check for Updates** in the JDeveloper menu to download the extension.; ii) You did not configure a keystore to sign the deployment. If so, then have a look at this website, …

http://docs.oracle.com/javase/tutorial/security/toolsign/step3.html

… create a keystore and a key entry and configure it for the Android platform in the JDeveloper **Tools -> Preferences -> ADF Mobile** menu.

Sample Project Overview

The **ViewController** project contains the a single **amx** page with the select many choice component. The **value** property of the select many choice component points to a managed bean that also exposes properties for the two input text components that display the index of the list selection and the name of the selected departments.



The managed bean code reads the selected list values from the value property to then access the ADF binding layer for the department name (or other row properties)

Note: The server side Web Service is accessed from a Web Service data control in the sample

Note: The code in this document is commented for you to learn what it does and how the "translated" list values are derived

```
import javax.el.ValueExpression;


import oracle.adfmf.bindings.dbf.AmxAttributeBinding;

import oracle.adfmf.bindings.dbf.AmxIteratorBinding;

import oracle.adfmf.bindings.iterator.BasicIterator;

import oracle.adfmf.framework.api.AdfmfJavaUtilities;

import oracle.adfmf.java.beans.PropertyChangeListener;
```

```
import oracle.adfmf.java.beans.PropertyChangeSupport;



public class SelectManyListChoiceHelper {
 //Property that holds the selectManyChoice list values. This property
 //is referenced from the af:selectManyChoice component's "value"
 //property
 Object[] selectedValues = null;

 //property that displayed the department names of the selected list
 //values
 String selectedValuesAsString = null;


 //to ensure that the two input text components are refreshed when the
 //list values are selected, we implement the change property change
 //notification that alerts the components to refresh as soon as their
 //value changes
 private transient PropertyChangeSupport propertyChangeSupport =
                                        new PropertyChangeSupport(this);
 public SelectManyListChoiceHelper() {
     super();
 }

 //note that none of the property change support needs to be manually
 //coded. When you generate accessors for the bean properties, just
 //ensure you click the checkbox for the property change support and
 //all the code gets created for you
 public void addPropertyChangeListener(PropertyChangeListener l) {
   propertyChangeSupport.addPropertyChangeListener(l);
 }


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

//method updated through the selectManyChoice component
public void setSelectedValues(Object[] selectedValues) {
    Object[] oldSelectedValues = this.selectedValues;
    this.selectedValues = selectedValues;
    propertyChangeSupport.firePropertyChange("selectedValues",
```

```
                                oldSelectedValues, selectedValues);

//force refresh of selected values field
propertyChangeSupport.firePropertyChange("selectedValuesAsString",
                "old", _getSelectedValueAsString(selectedValues));
}

public Object[] getSelectedValues() {
  return selectedValues;
}


//*** generated code ****
 public void setPropertyChangeSupport(
                     PropertyChangeSupport propertyChangeSupport) {
  PropertyChangeSupport oldPropertyChangeSupport =
                                          this.propertyChangeSupport;
  this.propertyChangeSupport = propertyChangeSupport;
  propertyChangeSupport.firePropertyChange("propertyChangeSupport",
                     oldPropertyChangeSupport, propertyChangeSupport);
 }

 public PropertyChangeSupport getPropertyChangeSupport() {
      return propertyChangeSupport;
 }

 //*** end of generated code ****


 //setter for the selectedValueAsString property. Note that this method
 //invokes a property change event that makes the UI component to
 //refresh
 public void setSelectedValuesAsString(String selectedValuesAsString) {
   String oldSelectedValuesAsString = this.selectedValuesAsString;
   this.selectedValuesAsString = selectedValuesAsString;
   propertyChangeSupport.firePropertyChange("selectedValuesAsString",
   oldSelectedValuesAsString, selectedValuesAsString);
}

public String getSelectedValuesAsString() {
  return _getSelectedValueAsString(selectedValues);
}

//THIS METHOD produces the department name based in the passed index
//values of the selected list items. So this is what you want to read
//and understand
private String _getSelectedValueAsString(Object[] selectedValues ) {
  selectedValuesAsString = "";
  if(selectedValues != null){
```

```
   //look up the list binding in the ADF PageDef file by its name
  //"DepartmentsMultiSelect".
     ValueExpression ve =
            (ValueExpression)AdfmfJavaUtilities.getValueExpression(

            "#{bindings.DepartmentsMultiSelect}",Object.class);
             AmxAttributeBinding attrBinding =
                           (AmxAttributeBinding)ve.getValue(
                            AdfmfJavaUtilities.getAdfELContext());

             //access the iterator that populates the list of values in
             //the selectManyChoice component

             AmxIteratorBinding amxListIterator =
                            attrBinding.getIteratorBinding();
             //the AmxIteratorBinding is a wrapper for the BasicIterator
             //iterator that exposes the information we need

             BasicIterator listIterator = amxListIterator.getIterator();
             //for each index value, query the department name (you can
             //access and attribute from the row) to display

             for(int i = 0; i<selectedValues.length; ++i){
               selectedValuesAsString = ""+selectedValuesAsString
                      +listIterator.getAttributeValueAtIndex(((new
                       Integer((String) selectedValues[i]))).intValue(),
                      "DepartmentName")+";";

             }
            return  selectedValuesAsString;
         }
      return "";
   }
}
```

Sample Download

You can download the JDeveloper 11.1.2.3 workspace with the sample as sample **"m08"**from the ADF Code Corner website

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

Follow the set-up instructions explained earlier before deploying it to your mobile device. Configuration includes

- providing a database connection to the HR schema

- Deployment of the HR WebService that is used to query data to the ADF mobile device

- Configure the connections.xml file with the IP address of the server that hosts the Web Service

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

⊠	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
⊠	