ADF Mobile Code Corner

m05. Caching WS queried data local for create, read, update with refresh from DB and offline capabilities

twitter.com/adfcodecorner

Abstract:

The current version of ADF Mobile supports three ADF data controls: Web Service, REST Service and POJO. When working with the WS and REST data control it is recommended practices to access the service throuh the WS and REST data control but to expose the data through the POJO data control to the UI.

The advantage of such an architecture is better control for filtering and manipulating WS queried data, as well as a better performance when working with data on the mobile device. In this article I explain how you query data from a SOAP Web Service, how you save the data in on device local entities and how to write local data back to the WS. In the provided sample, you find hookpoint to change for using a local SQLite database instead of an object cache for persistence.

Author: Frank Nimphius, Oracle Corporation
twitter.com/fnimphiu
25-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

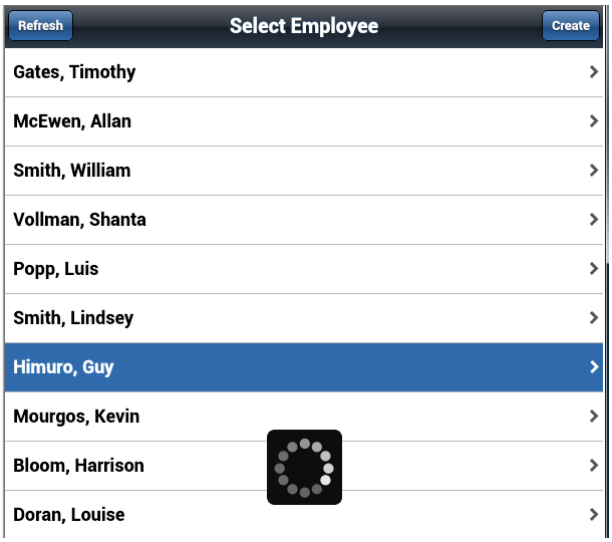
Introduction

For better performance when working with remote services in ADF Mobile, and to ensure data can be updated in offline mode, you want to look into how to cache queried data on the mobile device. This article explains how to use a POJO cache for this and how to access data queried from a WS data control in Java to expose in a POJO data control.

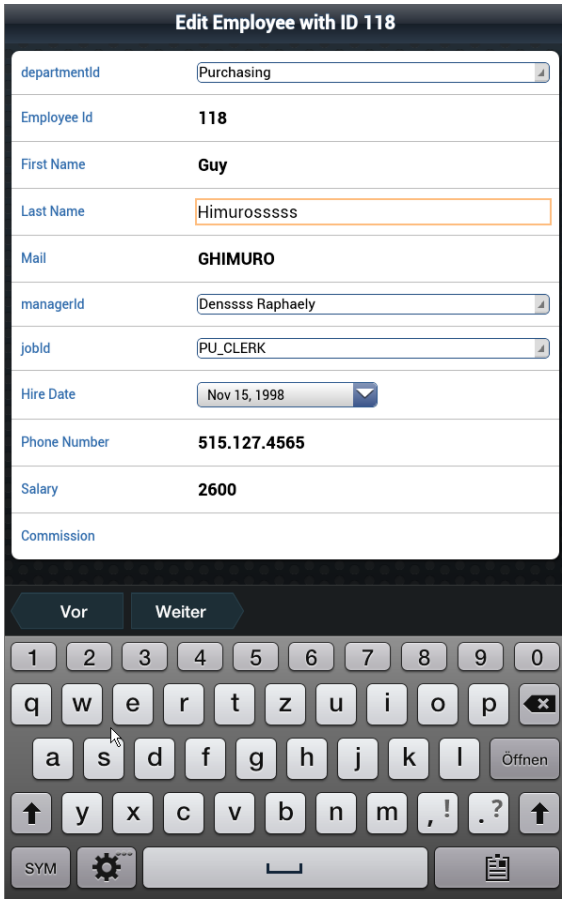
The use of the SQLite for offline data use is not in the scope of this article. However, writing and reading cached data to a local SQLite database for offline use and manual synchronization using the ADF Mobile lifecycle listener is quite easy to implement. You find sample code for accessing SQLite in the **HR** sample that is shipped with the ADF Mobile product. The sample for this ADF Code Corner article uses a POJO data control – `EmployeesListCache` – to dipatch between the server side Web Service and the mobile device for handling application read and update use cases. To integrate SQLite, you just hook into the read and write methods `getAllEmployees`, `getAllDepartment`, `getAllJobs` and `persistUpdatedEmployeeData`.

Note: After setting up your development environment you can examine ADF Mobile sample applications located in the `PublicSamples.zip` file in the `<jdev_install>/jdeveloper/jdev/extensions/oracle.adf.mobile/Samples` directory on your development computer. This includes the HR sample application.

At runtime, the sample application starts with a list view of employees queried through a SOAP service from the employees table of the Oracle HR database sample schema. Once queried, the employee data is cached in a POJO for local read and write operations. The POJO is exposed as a data control for declarative ADF UI-data binding.

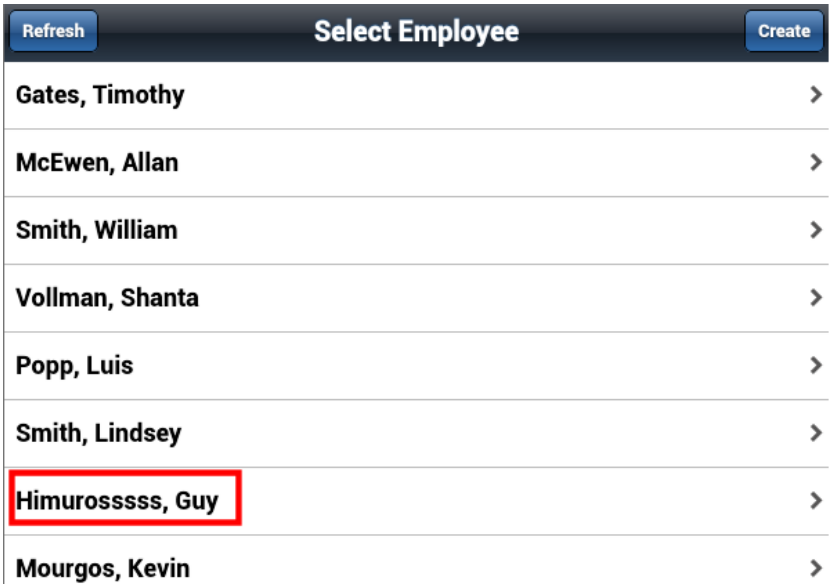


As shown in the image above, selecting a row in the list performs navigation to an edit page using the ADF task flow.



Changing data, as shown in the image above, and pressing **Submit** will persist the change in the local POJO and then call the Web Service to also persist the change in the database.

Note: Instead of persisting the change in the POJO and the database you could also use the SQLite database and defer server side persistence to a later point in time. This however is not part of the demo you can download for this article.



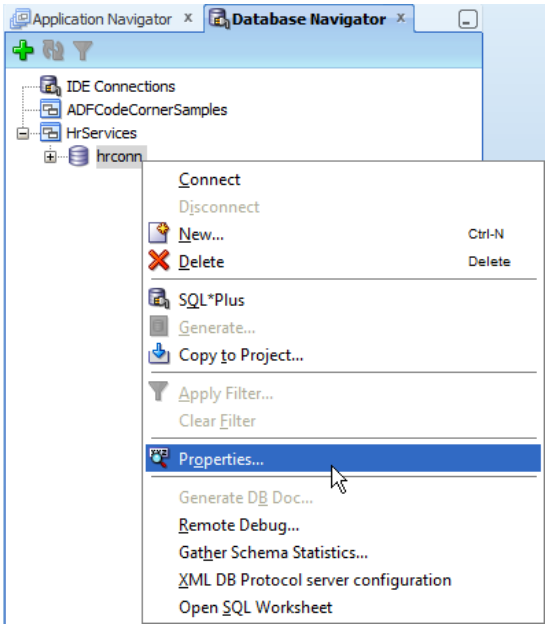
Pressing the **Refresh** button will delete the local POJO csache and re-query the data from the database so that changes added to the database are becoming visible on the mobile.

Configuring the Web Service for this Sample

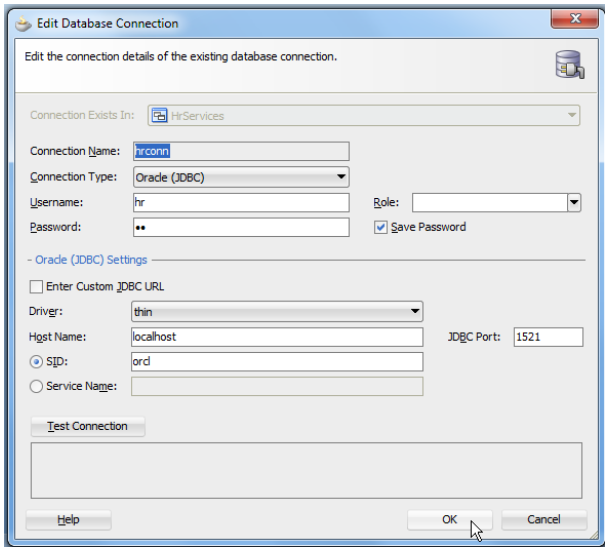
The ADF Code Corner mobile sample for this article queries data from the Oracle HR schema. The HR schema is accessed by an EJB model that is exposed as a SOAP Web Service. The Web Service project is provided as a separate JDeveloper 11.1.2.3 workspace so it can be deployed to the integrated WLS server.

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.

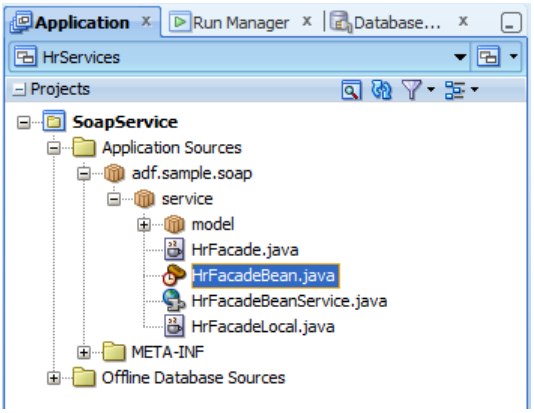


Choose **Properties** from the opened context menu to edit the database connect information as shown in the image below. Keep the name of the connection as **hrconn**.

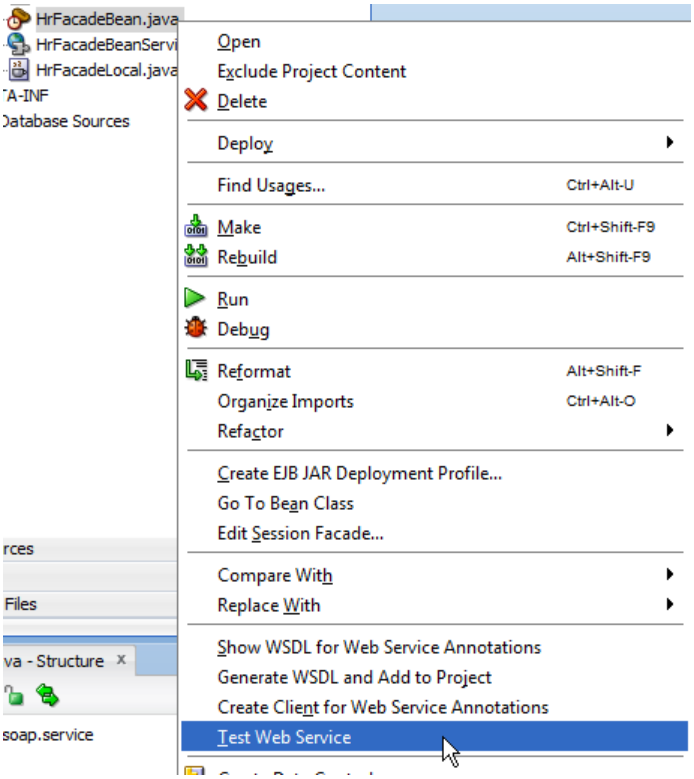


Use the **Test Connection** button to test the database connection. If the connection works, continue and run the Web Service in the JDeveloper WS tester.

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.

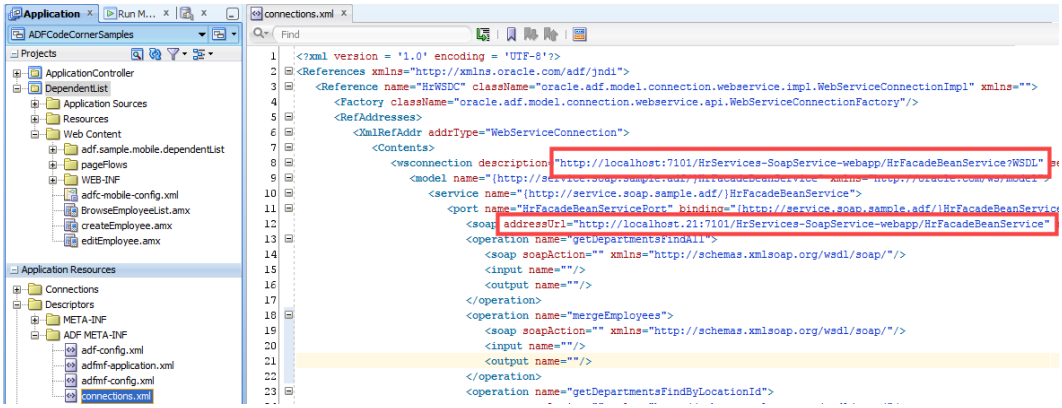


Note: If this is the first time you start the integrated WLS, a dialog shows for you to define a password for the integrated WLS administrator. Provide the password and keep all settings as defaulted. This way the integrated WLS also listens for the IP address or the domain address of your computer,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

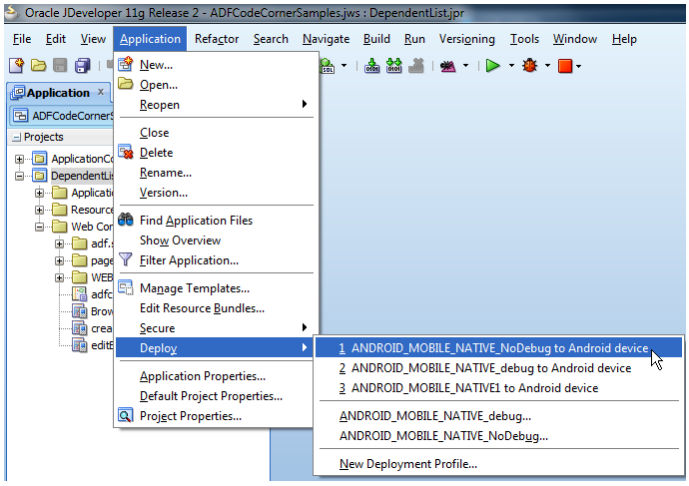
ADF Mobile Configuration & Deployment

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_NoDebug…** 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

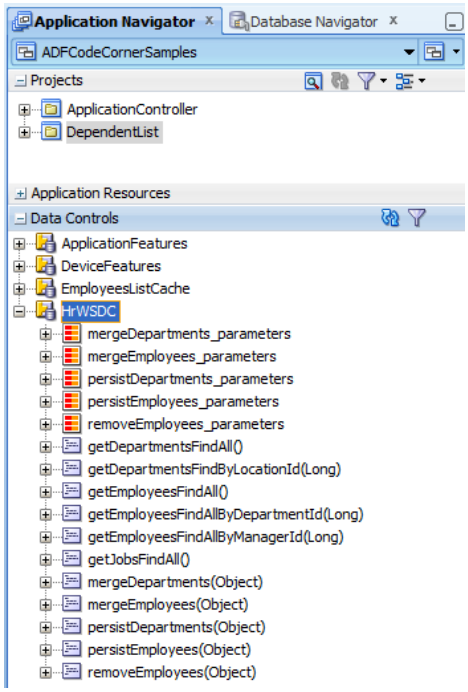
… create a keystore and a key entry and configure it for the Android platform in the JDeveloper **Tools -> Preferences -> ADF Mobile** menu.

POJO Caching Explained

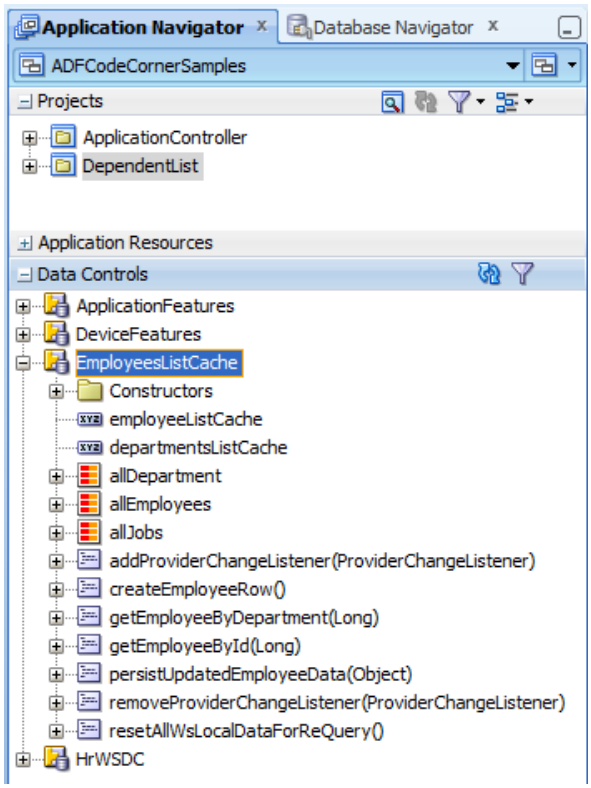
As mentioned already, the Web Service data is queried through an ADF Web Service Data Control, which is a purely declarative job to create at design time. The Web Service Data Control is then accessed from Java in a POJO - `EmployeesListCache` - that itself eposes collections read from the WS DC in a data control.

The Web Service Data Control

The Web Service Data Control is created in the **DependentList** view controller project and exposes all the method of the SOAP service. The WS Data Control is accessed from the `EmployeesListCache` class, which then also is exposed as a data control. The WS Data Control structure is shown in the image below.



The POJO data control is used in the ADF Mobile AMX page to bind the user interface components to data. As you can see, the names of the exposed methods and collections differ from the names in the WS Data Control, a clear indication that the POJO wraps the WS Data Control.



The Client Side POJO Model and WS DC access

The POJO accesses the Web Service Data Control using the following code lines

```
private HashMap _findAllEmployees() {

  try {
    //renew all employee data
    employeeListCache = new HashMap();

    //invoke WS data control from this POJO
      ArrayList parameterNames = new ArrayList();
      ArrayList parameterValue = new ArrayList()
      ArrayList parameterTypes = new ArrayList();

      //Generic representation of a data control's data provider
      //object. It exposes meta information about the providers
      //attributes and accessors, as well as the ability to get and set
      //their values.

      //This interface abstracts the raw data provider object from the
      //nature of its type of data control; the raw provider can be a
```

```
      //deserialized representation of an object returned by a SOAP or
      //REST web service call, or it can be an actual java class
      //instance. This interface is used primarily by the internals of
      //the embedded java data control framework, however it can also
      //be used when invoking data control methods directly from
      //application java bean code.

      GenericType employeesList =
              (GenericType)AdfmfJavaUtilities.invokeDataControlMethod(
               "HrWSDC", null, "getEmployeesFindAll", parameterNames,
               parameterValue, parameterTypes);

      if (employeesList != null) {
        for (int i = 0; i < employeesList.getAttributeCount(); i++) {
          GenericType row = (GenericType)employeesList.getAttribute(i);
          Employees employee =
              (Employees)GenericTypeBeanSerializationHelper
              .fromGenericType(Employees.class, row);
             employeeListCache.put(employee.getEmployeeId(),employee);

      }

    }

  } catch (AdfInvocationException e) {
     logApplicationMessage(Level.SEVERE, "Cannot read data from Web
                           Service \n", ""+ e.getMessage());
   }

  return employeeListCache;
}
```

Code Explained

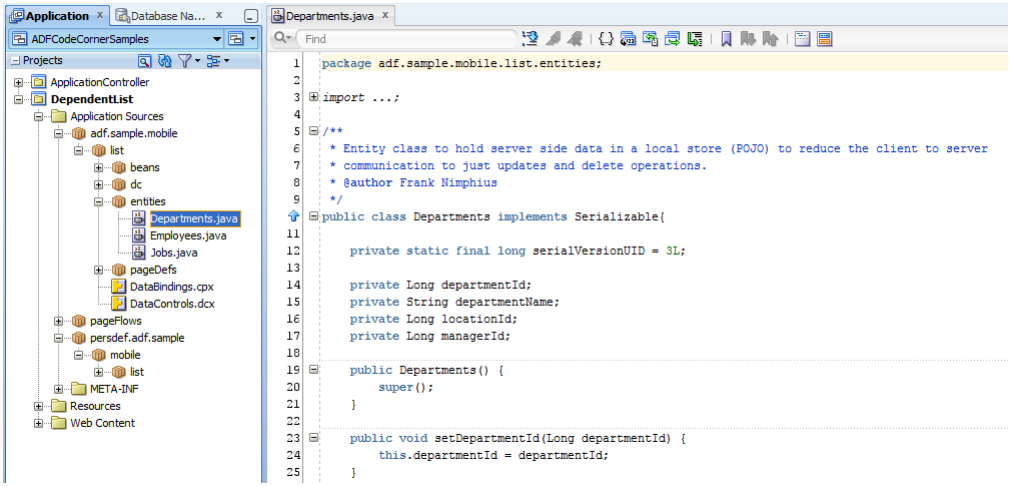
Note that the "**HrWSDC**" string is the ID of the Web Service Data Control like shown in the **DataControls.dcx** file or in the Data Control Window.

The **getEmployeesFindAll** method is the Web Service method that queries all employee data from the Oracle database. The getEmployeesFindAll method is exposed on the WS DC.

The Web Service method doesn't require any parameter argument and therefore empty **Lists** instances are passed in to this call.

The **GenericType** interface is explained later and and describes a generic representation of the data returned by the Web Service call. The GenericType contains access to all the WS row data attributes

The **Employees** entity is a client side modeling of the entity queried by the server side query. It's a POJO that contains all the attributes queried by the Web Service call and allows you to update the queried data locally. Similar, the DependentList project (the ViewController project in the sample) has entities for the departments (Departments) and jobs (Jobs) data defined. This way department and job information can be displayed in a LOV without the need to issue a query to the Web Service on each LOV call. The image below shows the entity class defined for the **Departments** data.



Note: To change the local cache from POJO to SQLite database, you would change the method above and use SQL commands to insert rows queried from the Web Service to the database. Of course, you can also use both, a POJO cache and the local database, in which case the local database would be used for caching data beyond application re-starts and for create, update and delete operations at runtime.

About GenericType

The `GenricType` interface is a generic data representation of the data control data provider object and exposes information about the provider object attributes and accessors. Though mainly used internally by the data controls in ADF Mobile, this interface can be used in custom mobile applications to model the structure of an object to update the server. The `GenericType` objects are only exposed for SOAP data controls. In this sample, `GenericType` is used to write employee data back to the server.

About GenericTypeBeanSerializationHelper

The `GenericTypeBeanSerializationHelper` helper class provides an easy way for application developers to convert between the ADF Mobile internal `GenericType` object and user defined POJOs.

Updating Server Side Data

The update of server side data (through the WS) is the reverse of the reading of remote data. Similar to the read access, the physical entities on the mobile device must be converted into a generic type, which then is passed as an argument to the Web Service Data Control method that updates the server side.

Note: The date field in the employee object caused problems which is why I manually created a new instance of `GenericVirtualType` instead of using one of the helper classes

```
/**
  * method to call out to the Web Service for persisting changes to
  * updated or new employee data. This method also changes the employee
  * data in this local POJO cache. Note that this sample does not check
  * if a newly created employee exists and instead just updates the
  * existing record. In a realistic use you would need to check if the
  * new object matches an existing object and alert the user, or use
```

```
* two different method for update and create persistence.
*
* @param emp employee object to persist
*/
public void persistUpdatedEmployeeData(Object emp) {
  //get Employees instance from ADF Mobile ConcreteJavaBeanObject
  //note that ConcreteJavaBeanObject is the object returned from the
  //ADF iterator as the current row using EL
  Employees employee = (Employees)
                       ((ConcreteJavaBeanObject)emp).getInstance();

  //update the local list with the new data
  if (employeeListCache == null) {
    employeeListCache = _findAllEmployees();
  }


  //check for null employee value in case of a new employee
  if(employee == null){
     AdfmfContainerUtilities.invokeContainerJavaScriptFunction
    ("adf.sample.mobile.list.dc.EmployeesListCache",
     "navigator.notification.alert",
      new Object[] {"Employee Object cannot be null." +
     " Update aborted.","", "Employee Object NULL", "Ok"});
     return;
  }
  boolean newEmployee = false;
  //check wthere update or new object creation needs to be performed.
  //If the employee object is not found in the local cache then we
  //assume it is a new object that needs to be persisted on the
  //server and added to the local cache.
  newEmployee = this.getEmployeeById(
         employee.getEmployeeId()).length == 0? true : false;

  try{
      ArrayList parameterNames = new ArrayList();
      parameterNames.add("arg0");
     //create a new generic type. Note that the order of attributes
     //must match the order in the Web Service


     //Define a metadata object model of the entity structure
     GenericVirtualType gvt = new GenericVirtualType();
     gvt.defineAttribute("http://schemas.xmlsoap.org/wsdl/soap/",
                         "commissionPct",Double.class);
```

```
gvt.defineAttribute("http://schemas.xmlsoap.org/wsdl/soap/",
                    "departmentId",Long.class);
gvt.defineAttribute("http://schemas.xmlsoap.org/wsdl/soap/",
                    "email", String.class);
gvt.defineAttribute("http://schemas.xmlsoap.org/wsdl/soap/",
                    "employeeId",Long.class);
gvt.defineAttribute("http://schemas.xmlsoap.org/wsdl/soap/",
                    "firstName", String.class);
gvt.defineAttribute("http://schemas.xmlsoap.org/wsdl/soap/",
                    "hireDate", Date.class);
gvt.defineAttribute("http://schemas.xmlsoap.org/wsdl/soap/",
                    "jobId", String.class);
gvt.defineAttribute("http://schemas.xmlsoap.org/wsdl/soap/",
                    "lastName", String.class);
gvt.defineAttribute("http://schemas.xmlsoap.org/wsdl/soap/",
                    "managerId",Long.class);
gvt.defineAttribute("http://schemas.xmlsoap.org/wsdl/soap/",
                    "phoneNumber", String.class);
gvt.defineAttribute("http://schemas.xmlsoap.org/wsdl/soap/",
                    "salary",Double.class);


//copy values of the updated row object into a generic type to
//post to the Web Service

gvt.setAttribute("commissionPct",employee.getCommissionPct());
gvt.setAttribute("departmentId",employee.getDepartmentId());
gvt.setAttribute("email", employee.getEmail());
gvt.setAttribute("employeeId",employee.getEmployeeId());
gvt.setAttribute("firstName", employee.getFirstName());
gvt.setAttribute("hireDate", employee.getHireDate());
gvt.setAttribute("jobId", employee.getJobId());
gvt.setAttribute("lastName",employee.getLastName());
gvt.setAttribute("managerId",employee.getManagerId());
gvt.setAttribute("phoneNumber",employee.getPhoneNumber());
gvt.setAttribute("salary",employee.getSalary());

//Create the input parameters to pass to the web service
//operation

ArrayList parameterValue = new ArrayList();
parameterValue.add(gvt);
ArrayList parameterTypes = new ArrayList();
parameterTypes.add(Object.class);
```

```
        //WS is based on EJB and thus we do have a merge operation for
        //existing objects and a persist operation for new objects. We
        //use the "newEmployee" flag to distinguish the two use cases
      if (newEmployee) {
         AdfmfJavaUtilities.invokeDataControlMethod("HrWSDC", null,
                                "persistEmployees", parameterNames,
                                 parameterValue,parameterTypes);
      //add new object t local cache
      employeeListCache.put(employee.getEmployeeId(), employee);
      //notify update
      this.setEmployeeListCache(employeeListCache);
      //notify container about the create
      } else {
         AdfmfJavaUtilities.invokeDataControlMethod("HrWSDC", null,
                                "mergeEmployees", parameterNames,
                                   parameterValue,parameterTypes);
        // replace current entry in local cache
        employeeListCache.put(employee.getEmployeeId(), employee);
        //notify update
        this.setEmployeeListCache(employeeListCache);
      }
   } catch (AdfInvocationException e) {
        logApplicationMessage(Level.SEVERE, "Cannot update data
                    accessing Web Service \n", ""+ e.getMessage());
   }
}
```

Sample Download

There is a lot more to say about this sample and therefore you want to have a look at its functionality and implementation after downloading the sample zip file. The Oracle JDeveloper 11.1.2.3 sources can be downloaded as sample "m05" from the ADF Code Corner website:

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

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

⊠	
⊠	Class GenericTypeBeanSerializationHelper http://docs.oracle.com/cd/E38668_01/apirefs.111230/e27204/oracle/adfmf/framework/api/GenericTypeBeanSerializationHelper.html
⊠	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