# ORACLE ADF ON-RAMP: DEVELOPING APPLICATIONS WITH THE CORE ADF STACK

*Peter Koletzke, Quovera*

*Si nous ne trouvons pas des choses agréables, nous trouverons du moins des choses nouvelles.*

(If we do not find anything pleasant,
at least we shall find something new.)

—Voltaire (1694-1778), *Candide* (Ch. xvii)

Developing a Java-oriented web application these days is an experience that many Oracle technologists find to be new but not necessarily very pleasant. Architecting such an application requires selecting a set of technologies from a dauntingly-large and ever-growing list. Up to now, the responsibility for combining these technologies and ensuring that they communicate and work together in an orderly way has been left up to each organization. The path of selecting and working with different Java-oriented frameworks can inevitably lead to wrong turns, especially for those who are new to the Java world. Depending upon when those wrong turns occur, the effect on the project can range from mild to devastating and will likely require rewriting some or most of the application.

Fortunately, Oracle has now provided guidance in the form of the set of technologies they have selected to build Fusion Applications—the new application products (parallel to E-Business Suite). Oracle has chosen open-standards technologies in the Java realm so parts of the application can be easily extended with little reliance on a specific vendor's product line, hardware set, or operating system. (Many other reasons for selecting open standard technologies—such as customer preferences—exist, but are a larger discussion that is not critical to the focus of this white paper.)

## Fusion Technology Stack

Fusion developers within Oracle have been creating Fusion Applications using Application Development Framework (ADF) in JDeveloper with the following core technologies:

- ADF Business Components (ADF BC)
- ADF Faces Rich Client (ADF Faces RC)
- ADF Bindings and ADF Data Controls
- ADF Controller

In addition to those core technologies, Oracle uses high-level technologies or strategies such as the following to coordinate Fusion Applications' components and to fulfill additional architectural requirements:

- Service Oriented Architecture (SOA) with Business Process Execution Language (BPEL)
- Enterprise Service Bus (ESB)
- Oracle Business Rules
- Oracle WebCenter

Since Oracle is using Application Development Framework (ADF)—a facility in JDeveloper for working with code in a common way—to create Fusion Applications, you can use the term "ADF Fusion Technology Stack" to refer to all technologies in the core and high-level lists. Packaged application software is a large part of Oracle's business, and Oracle has a very compelling business reason to ensure that the technologies used in Fusion Applications will integrate properly and work successfully. Therefore, you can be relatively assured that you, too, can be successful in creating applications with the same technologies.

## Retooling for Fusion Technology Work

Determining the list of technologies that an application will use is not enough. Planning for any application development effort must also include tasks and strategies for bringing current development staff up to speed on the techniques required for the new environment. If you determine that your current development staff cannot reach acceptable skill levels in the available time, you may need to employ additional resources. You will need to understand what tools, development techniques, and languages a developer needs to learn (for current staff) or to know (for additional resources) to be productive in the ADF Fusion Technology environment.

The main objective of this white paper is to explain just that—what developers need to know to be productive writing applications using the ADF Fusion Technology Stack. If you think of Oracle internal developers as drivers already speeding along on the Fusion Development Highway, this white paper is the on-ramp for others who are not yet on that road but who need to be there. To extend that analogy a bit, while the exit for Oracle developers is "Oracle Fusion Applications Production" and yours will be different, all have the same vehicle—ADF in JDeveloper—and type of fuel—the Fusion Technology Stack.

This white paper starts by explaining some preliminary concepts; then it explains and shows the kinds of code and techniques needed for productive work in ADF with the core technologies in the Fusion Technology Stack. The goal is to explain the main development techniques for only the core technology set. The high-level technologies are more strategic systems that an enterprise architect will select for a particular application. While heads-down developers may need to know about techniques for the high-level technologies, that type of work will vary depending upon architectural decisions and on the enterprise's environment. In fact, the core technology stack may suffice for some applications so no high-level technologies would be needed at all. The white paper closes by discussing the languages developers use for this type of work.

**Note**: Although this white paper does not specifically address techniques required to extend Fusion Applications, if you currently develop or maintain custom extensions to Oracle E-Business Suite (Oracle Applications) or think you will find yourself doing so in the future, you will be using the same techniques and technologies discussed in this white paper for that work in Fusion Applications.

## What is Fusion?

The word "Fusion" is used these days to refer to almost everything from cars to food to razors to drinks. Therefore, the first concept to understand is what Oracle means when they use that word. The word "Fusion" is used in various ways within the Oracle product line, but it generally refers to a strategic reorganization of Oracle products. Oracle invented Oracle Fusion after acquiring various companies who offered their own application products. Oracle's objective with Fusion is to merge the best of all those products into a single (fused) applications suite. This effort will take many years, but Oracle has started this work and we expect to see the premier version of these application products in the near future.

You can summarize Oracle Fusion with the uses in the following terms:

- **Oracle Fusion Applications**, mentioned before, is the next version of Oracle E-Business Suite.
- **Oracle Fusion Middleware** is the toolset that Oracle is using to develop and deploy Fusion Applications. This toolset consists of virtually all Oracle development and runtime products (except for the Oracle database and Oracle packaged applications) such as JDeveloper and Oracle WebLogic Server (but not legacy tools such as Oracle Forms and Reports).
- **Oracle Fusion Architecture** outlines the way various technologies are used to build the applications. This Fusion usage is not as frequently used or seen as Oracle Fusion Middleware and Oracle Fusion Applications.

### JDeveloper and ADF

JDeveloper 11*g* is the Fusion Middleware development tool. It is the common tool used for developing all types of code, regardless of the technology. Moreover, as mentioned, JDeveloper is the container for ADF. Therefore, Oracle is very focused on enabling JDeveloper 11*g* to support all requirements of the new Fusion Applications.

Although this white paper focusses on JDeveloper as the main tool to use for ADF, Oracle has published *ADF Essentials*, a package of no-license-fee ADF technologies that you can also plug into Eclipse through the Oracle Enterprise Pack for Eclipse. Instead of WebLogic Server, the public domain server Glassfish is used for deployment and runtime of ADF Essentials. More information is available on the Oracle ADF web pages with a good starting point being the FAQs: www.oracle.com/technetwork/developer-tools/adf/overview/adfessentialsfaq-1837249.pdf.

## What is ADF?

To answer that question, you need to know that the word "framework" in the Java world refers to an application development technology. A framework is like an Application Programming Interface (API) or a code library in other disciplines: all offer generically built code that you can use in your application. The code that implements the framework supplies an entire service that you can access using a certain development method and calling interface. Although APIs and code libraries may have these characteristics, frameworks are built around the idea of a service. For example, instead of building from scratch some key facility such as a connection layer to the database, you use an existing framework such as ADF BC to supply that service.

One reason to use a framework is to tap into a standard way of supplying the functionality of the service to your application. You do not need to invent a service that you need for a piece of your application. Another related reason is that you do not need to redevelop code that many applications share. When using a framework, you leverage solid and (hopefully) well-debugged code in all your applications. In addition, the most popular frameworks offer solid support at least from the user community, if not from a vendor. The sidebar "Working with Java Frameworks" describes how you use frameworks in your application code.

**Working with Java Frameworks**

Framework code in the Java world usually consists of prebuilt Java classes. Those classes offer complete functionality for a service (like database access). They are set up to read configuration or application-specific definitions coded inside an Extensible Markup Language (XML) file. Therefore, the primary code you are responsible for when using a framework is XML-based. A good framework offers enough flexibility to handle most applications with this type of work. Moreover, developers using frameworks are most effective when they understand what the framework can accomplish so they can design their application code to fully leverage the framework.

Occasionally (and if you are using frameworks properly, it should only be occasionally), a developer will need to replace or add to a part of the service that cannot fulfill an application's requirement. In this case, the developer subclasses one or more framework classes and adds some code to customize the framework's behavior. This type of work requires intermediate-level knowledge of Java as well as a deep knowledge of the framework. Therefore, it is a technique to be used sparingly.

### So ADF is…

*Application Development Framework* (ADF) is an architectural strategy within JDeveloper that allows you to build applications using common declarative and visual methods. For example, you can build database access code into your application using Enterprise JavaBeans (EJBs), ADF Business Components (ADF BC), or web services (among others). The code details and libraries that support these frameworks are different, but the actions you use in JDeveloper to create user interfaces based on these frameworks are the same. ADF, therefore, is really a meta-framework that integrates and offers common development methods to many other frameworks.

## ADF Architecture

The ADF architecture model, depicted in Figure 1, divides the frameworks it supports into various code layers that loosely follow the Java EE design pattern Model-View-Controller (MVC). *MVC* defines three main layers of application code: *Model*—to manage the data portion of the application, *View*—to handle drawing the user interface screen, and *Controller*—to process user interface events (such as button clicks) and to control page flow (how one page is called from another page).

The ADF architecture layers follow the definition of MVC for the most part, but ADF adds another layer, *ADF Business Services*, a spin off from the Model layer. ADF Business Services provides code for accessing data sources such as a database. Business services are responsible for *persistence*—the physical storage of data for future retrieval—and *object-relational (OR) mapping*—translating storage units such as rows and columns in relational database tables to object-oriented structures such as arrays of objects with property values. ADF Business Components is a core Fusion technology in this layer.

The *ADF View* layer corresponds directly to the MVC View layer. It includes technologies that you use to draw the user interface. In the case of *web client code*—application code that is run in a Java runtime on an application server rather than locally on the desktop (as is *application client code*)—ADF View supports JavaServer Faces (JSF) and ADF Faces RC, core Fusion technologies.

The *ADF Controller* layer, which defines separate frameworks only for web client code, supports popular JSF and Struts controller frameworks. In addition, it adds an ADF-specific framework—ADF Controller ("ADF Task Flow Controller") — that allows you to create and control parts of a page. ADF Controller is a core Fusion technology in this layer.

The *ADF Model* layer corresponds to part of the MVC Model layer but specifically represents the connection mechanism from the Business Services layer to the View layer (through the Controller layer). The ADF Model layer is composed of the following two aspects:

- **ADF Bindings** This framework (really just an aspect of ADF Model) provides a standard way to access data values in the ADF Business Services layer from an ADF View user interface component such as a pulldown item. For example, if you defined a business service item to query the DEPARTMENTS table, you could add an expression to the *Value* attribute of a text input item referring to the DEPARTMENT_ID column of the query. When the screen is drawn, the data would automatically flow from the ADF Business Services object to the text item in the View layer by using ADF Bindings.
- **ADF Data Controls** This aspect of ADF Model supplies a list of prebound components based on the data model (data sources) defined in the ADF Business Services layer. For example, in JDeveloper, you could drag and drop a node from the Data Controls panel that represents the DEPARTMENTS query onto a JSF page. The IDE will determine the type of business service (in this case a *collection*—multiple rows and multiple columns) and will present a selection menu of different styles of display components (for example, forms, tables, trees, or navigation buttons). Selecting one of those options causes JDeveloper to lay out the appropriate display on the screen and bind the items on the screen to the business service.

Using both of those aspects, you do not need to write code to present data (for query and also for insert, update, and delete operations) in the user interface. Although no Java EE standard exists yet for bindings and data controls, Oracle and other parties were working on a *Java Specification Request* (JSR, the process by which a new feature or revision is made to the Java platform) to include this mechanism in the Java standards. (This JSR was removed in May 2011 but you can review its history by searching at jcp.org for JSR-227.)

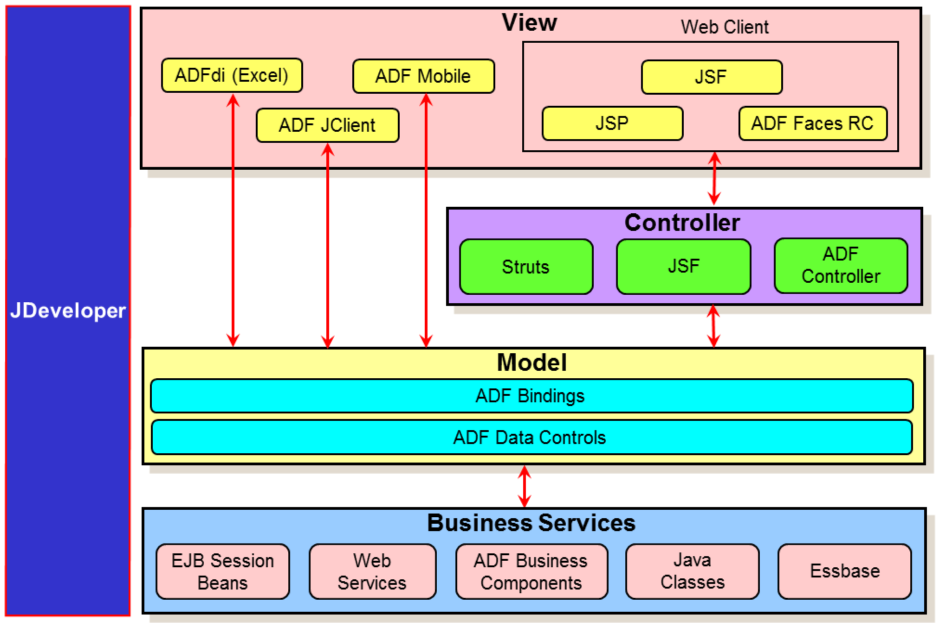


**Figure 1. ADF architecture model**

You will also notice in Figure 1 that JDeveloper sits to the side of the ADF framework layers because it is the tool you use to manipulate all ADF technologies.

*Dans ce meilleur des mondes possibles ...*
*tout est au mieux.*

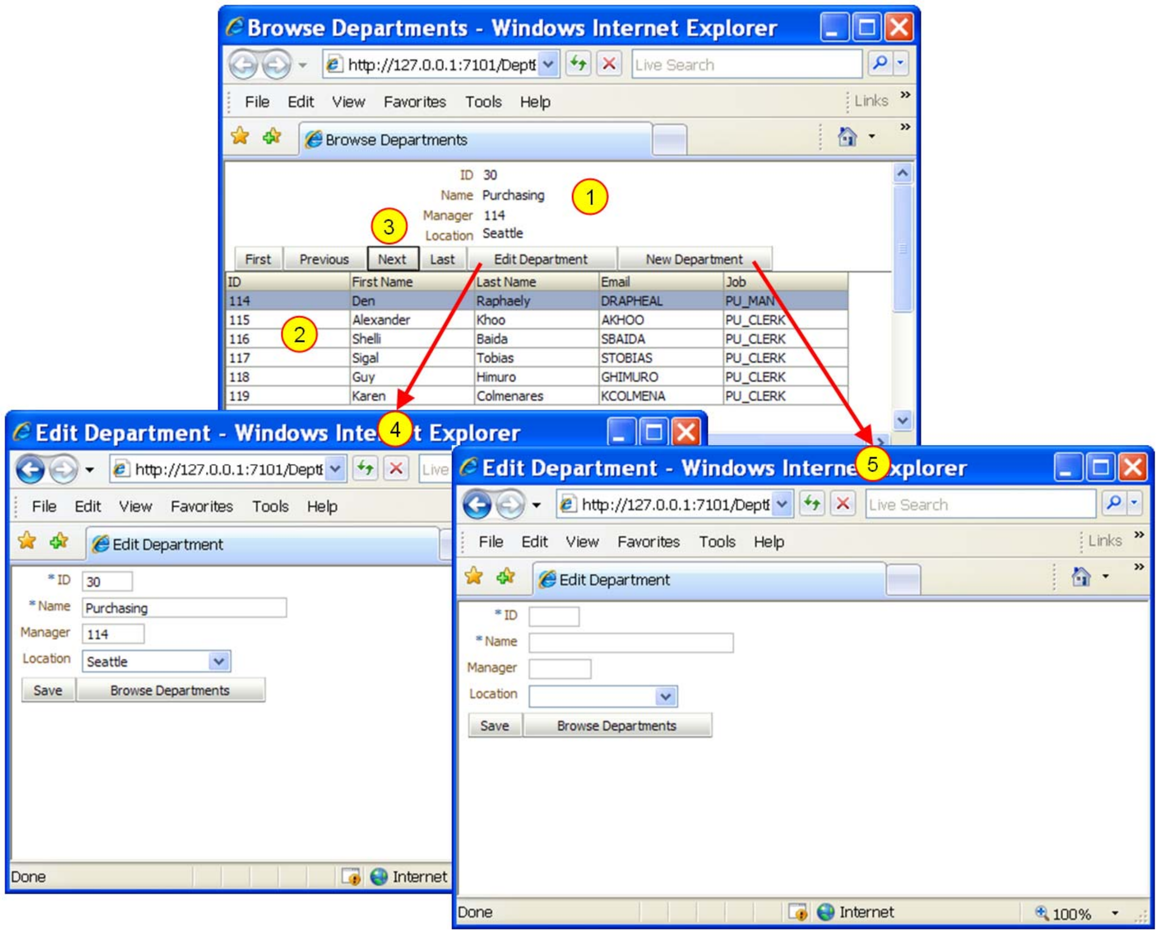
(In this best of all possible worlds ...
everything is for the best.)

—Voltaire *(*1694-1778), *Candide* (Ch. i)

## Core ADF Fusion Technologies

The easiest way to describe the core ADF Fusion technologies is in the context of a working application. Although the ADF frameworks have many advanced features, the purpose of this white paper (to understand what you need to know) will be best served by looking at a simple application (shown in Figure 2) that provides the following basic data handling functions:

1. Querying the DEPARTMENTS table in read-only mode when the page opens.
2. Querying EMPLOYEES table records that are related to the displayed DEPARTMENTS record.
3. Navigating between DEPARTMENTS table records using **First**, **Previous**, **Next**, and **Last** buttons.
4. Editing the displayed DEPARTMENTS table using a separate page accessed with the **Edit Department** button.
5. Creating a DEPARTMENTS record using the edit page in Create mode accessed with the **New Department** button.


Browse Departments - Windows Internet Explorer
http://127.0.0.1:7101/Dept
Live Search
File Edit View Favorites Tools Help
Links
Browse Departments
1
2
3
First Previous Next Last Edit Department New Department
Edit Department - Windows Internet Explorer
4
5
Save Browse Departments
Done
Internet
100%

| | |
|---|---|
| ID | 30 |
| Name | Purchasing |
| Manager | 114 |
| Location | Seattle |

| ID | First Name | Last Name | Email | Job |
|---|---|---|---|---|
| 114 | Den | Raphaely | DRAPHEAL | PU_MAN |
| 115 | Alexander | Khoo | AKHOO | PU_CLERK |
| 116 | Shelli | Baida | SBAIDA | PU_CLERK |
| 117 | Sigal | Tobias | STOBIAS | PU_CLERK |
| 118 | Guy | Himuro | GHIMURO | PU_CLERK |
| 119 | Karen | Colmenares | KCOLMENA | PU_CLERK |

| | |
|---|---|
| * ID | 30 |
| * Name | Purchasing |
| Manager | 114 |
| Location | Seattle |

| | |
|---|---|
| * ID | |
| * Name | |
| Manager | |
| Location | |

**Figure 2. Sample application containing basic data handling functions**

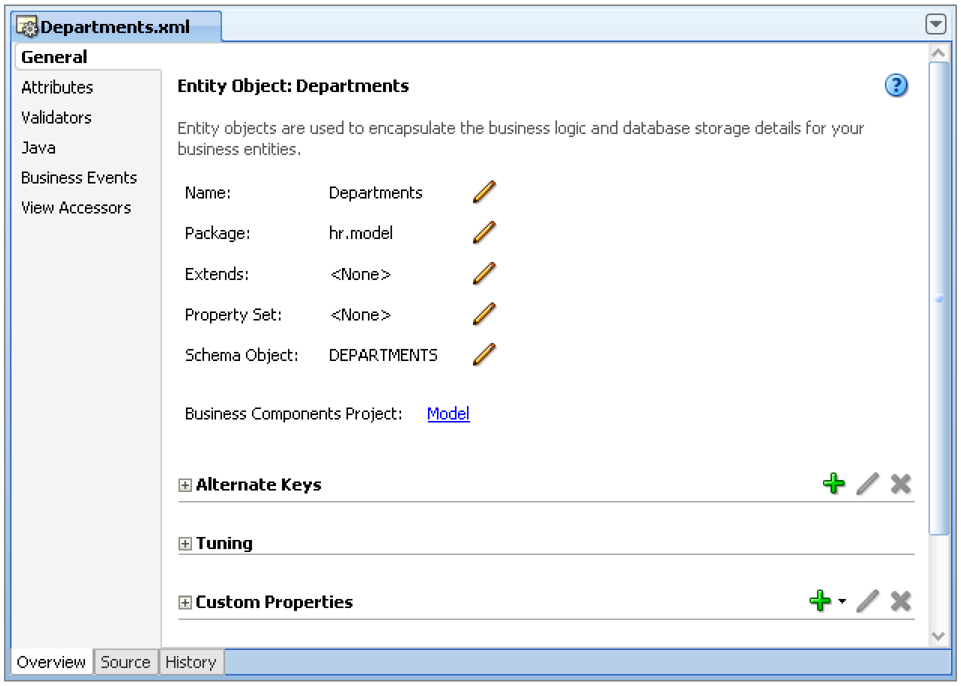
This application uses basic examples of these core Fusion technologies:

- **ADF Business Components** for ADF Business Services layer functions that access the database.
- **ADF Faces Rich Client** for ADF View layer functions that render the user interface in the web browser
- **ADF Bindings and ADF Data Controls** for ADF Model layer functions that connect database data to components on the web page
- **ADF Controller** for Controller layer functions that manage page flow and handle user event interactions

Let's see where those technologies are used in this sample application.

## ADF Business Components

This application queries and updates data in an Oracle database. ADF Business Components (ADF BC) is the framework from the ADF Business Services layer used to perform the database-specific operations. For example, a representation of the DEPARTMENTS table is defined in an ADF BC *entity object*. You work with the entity object code in a declarative way. When you create an entity object, you follow a set of wizard pages. To change the entity object you would interact with a property editor such as the following for the Departments entity object:



Entity Object: Departments

Entity objects are used to encapsulate the business logic and database storage details for your business entities.

| | |
|---|---|
| Name: | Departments |
| Package: | hr.model |
| Extends: | <None> |
| Property Set: | <None> |
| Schema Object: | DEPARTMENTS |

Business Components Project: Model

Entity objects contain *attributes* that represent columns in the database table or view. The Attributes tab in the Entity Object Editor just shown allows you to modify the details about a specific entity attribute. Figure 3 shows an example of that screen.

Each attribute defines a Java field (for example, DepartmentId with a Java type of Number and a SQL type of NUMBER(4,0)) that ADF BC will use to prepare INSERT, UPDATE, and DELETE statements based on instructions issued through the user interface. These SQL statements are then passed to the database through Java Database Connectivity (JDBC) communication paths. All of the code that handles the JDBC calls as well as the code to create the SQL statements are provided by ADF BC. All you need do is declare at which table and columns the ADF BC framework should target.

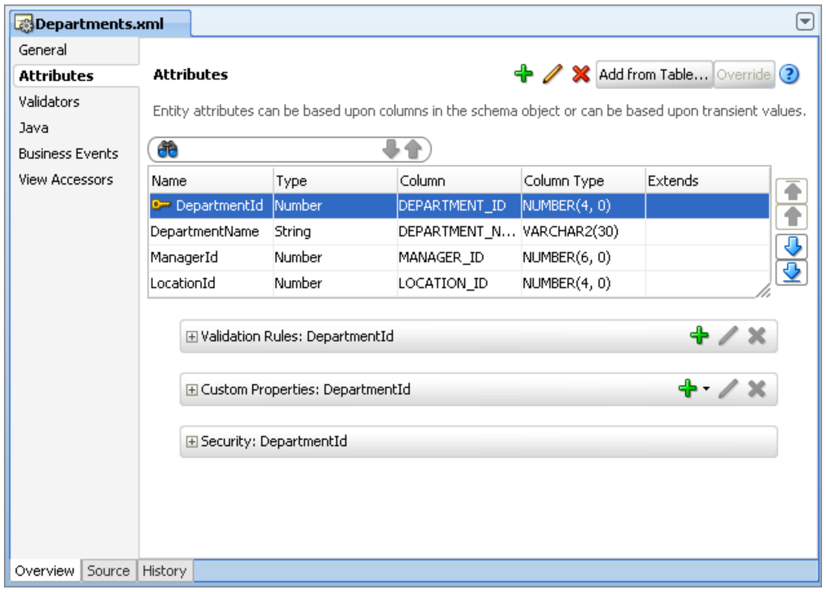
The entity object wizard pages and property editor screens create XML code that is read by the framework files. The following code listing is a snippet from Departments.xml, the entity object definition file for the DEPARTMENTS table:

```
<Entity
  xmlns="http://xmlns.oracle.com/bc4j"
  Name="Departments"
  Version="11.1.1.53.41"
  DBObjectType="table"
  DBObjectName="DEPARTMENTS"
  AliasName="Departments"
```

```
    BindingStyle="OracleName"
    UseGlueCode="false">
    <DesignTime>
      <Attr Name="_codeGenFlag2" Value="Access"/>
      <AttrArray Name="_publishEvents"/>
    </DesignTime>
    <Attribute
      Name="DepartmentId"
      IsNotNull="true"
      Precision="4"
      Scale="0"
      ColumnName="DEPARTMENT_ID"
      SQLType="NUMERIC"
      Type="oracle.jbo.domain.Number"
      ColumnType="NUMBER"
      TableName="DEPARTMENTS"
      PrimaryKey="true">
      <DesignTime>
        <Attr Name="_DisplaySize" Value="22"/>
      </DesignTime>
      <Properties>
        <SchemaBasedProperties>
          <LABEL
            ResId="hr.model.Departments.DepartmentId_LABEL"/>
        </SchemaBasedProperties>
      </Properties>
    </Attribute>
```

This snippet shows how the entity object is declared and associated with the DEPARTMENTS table; it also sets up the DepartmentId attribute based on the DEPARTMENT_ID column. Similar definitions appear for other attributes in the entity object. When you change the entity object properties, the XML code is modified appropriately. Therefore, you do not need to modify (or even look at) entity object XML code.

**Note**: This declarative style of programming is found throughout work in JDeveloper and is a core strength of ADF.



**Attributes**

Entity attributes can be based upon columns in the schema object or can be based upon transient values.

| Name | Type | Column | Column Type | Extends |
|---|---|---|---|---|
| DepartmentId | Number | DEPARTMENT_ID | NUMBER(4, 0) | |
| DepartmentName | String | DEPARTMENT_N... | VARCHAR2(30) | |
| ManagerId | Number | MANAGER_ID | NUMBER(6, 0) | |
| LocationId | Number | LOCATION_ID | NUMBER(4, 0) | |

Validation Rules: DepartmentId

Custom Properties: DepartmentId

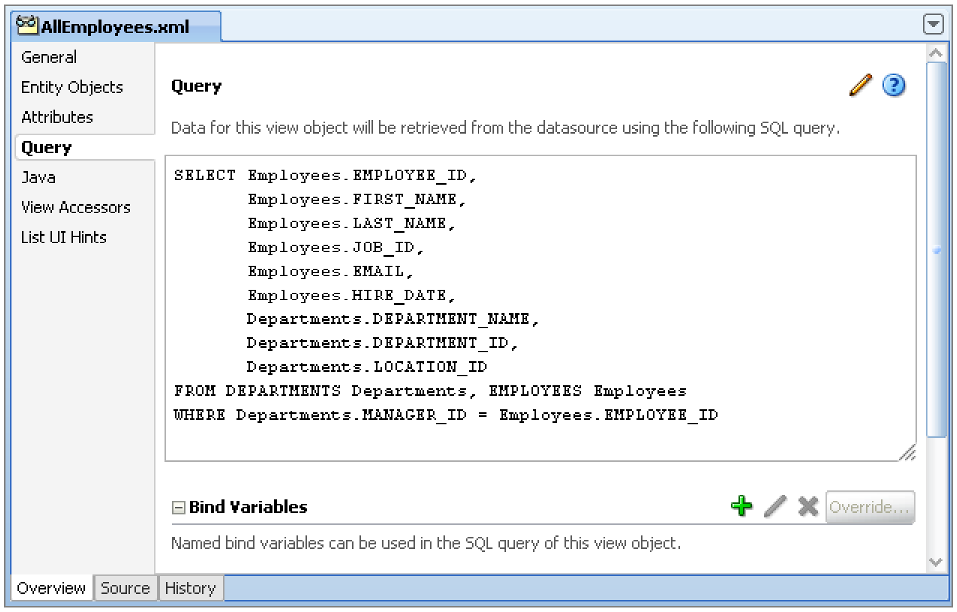
Security: DepartmentId

**Figure 3. Attributes page of the Entity Object Editor**

Just as entity objects supply INSERT, UPDATE, and DELETE operations, *view objects* represent SELECT statements. View objects can be based on one or more entity objects, which then supply details about the table and columns, or on SELECT statements. You create and edit view objects in the same declarative way as entity objects. An XML code snippet for a view object follows.

```
<ViewObject
  xmlns="http://xmlns.oracle.com/bc4j"
  Name="AllEmployees"
  Version="11.1.1.53.41"
  SelectList="Employees.EMPLOYEE_ID,
       Employees.FIRST_NAME,
       Employees.LAST_NAME,
       Employees.JOB_ID,
       Employees.EMAIL,
       Employees.HIRE_DATE,
       Departments.DEPARTMENT_NAME,
       Departments.DEPARTMENT_ID,
       Departments.LOCATION_ID"
  FromList="DEPARTMENTS Departments, EMPLOYEES Employees"
  Where="Departments.MANAGER_ID = Employees.EMPLOYEE_ID"
  BindingStyle="OracleName"
  CustomQuery="false"
  PageIterMode="Full"
  UseGlueCode="false">
  ...
   <ViewAttribute
    Name="EmployeeId"
    IsUpdateable="false"
    IsNotNull="true"
    PrecisionRule="true"
    EntityAttrName="EmployeeId"
    EntityUsage="Employees"
    AliasName="EMPLOYEE_ID"/>
```

This view object is based on two entity objects, Employees and Departments; in the view object's XML you will find clauses used to construct a SELECT statement from those two tables. You can also read this query more directly in the view object editor as shown here:



Query

Data for this view object will be retrieved from the datasource using the following SQL query.

```
SELECT Employees.EMPLOYEE_ID,
       Employees.FIRST_NAME,
       Employees.LAST_NAME,
       Employees.JOB_ID,
       Employees.EMAIL,
       Employees.HIRE_DATE,
       Departments.DEPARTMENT_NAME,
       Departments.DEPARTMENT_ID,
       Departments.LOCATION_ID
FROM DEPARTMENTS Departments, EMPLOYEES Employees
WHERE Departments.MANAGER_ID = Employees.EMPLOYEE_ID
```

Bind Variables

Named bind variables can be used in the SQL query of this view object.

The Bind Variables section of the editor just shown allows you to create variables that you work into the query so you can filter rows by values supplied by the application or by the user.

You can also create *view links* that represent foreign key constraints, master-detail relationships, or other logical attribute pairs that relate one view object to another. In the sample application, a view link is defined between the DepartmentsView and EmployeesView view objects so when a department record is displayed, the employees for that department will be displayed. ADF BC automatically handles the master-detail synchronization if you define a view link.

## ADF Controller

The JavaServer Faces standard of the Java Enterprise Edition platform specifications defines Controller functionality, which manages page flow (which page is loaded) as well handling user events (for example, by passing data from the Model layer to the View layer). ADF supplements the standard JSF Controller with the ADF Controller framework (also called "ADF Task Flow Controller"), which adds the ability to handle *page fragments* (parts of pages).

This ability has the following advantages over the standard JSF Controller:

- Page fragment processing can be faster (because fewer components are rerendered)
- Fragments can be reused more easily than full pages
- Additional functions or logic can be added into the flow between pages
- Flows between pages can be reused in different parts of the application.

The sample application does not specifically demonstrate page fragments; instead, as a simpler example, it shows a more standard set of two full pages: browse and edit. Navigating from one to the other is handled by the Controller as is the activity triggered by button clicks—for example, the **Next** and **Previous** buttons. Defining page flow is easiest using the diagrammer shown in Figure 4. You first create an ADF Controller file, and then drop View (page) and Control Flow Case (flow) components onto it. You then name all objects so you can refer to them in code later on. As with ADF BC, when you interact with the diagram editor, JDeveloper creates XML code such as the following:

```
<task-flow-definition id="dept-flow">
  <default-activity>deptBrowse</default-activity>
  <view id="deptBrowse">
    <page>/deptBrowse.jspx</page>
  </view>
  <view id="deptEdit">
    <page>/deptEdit.jspx</page>
  </view>
  <control-flow-rule>
    <from-activity-id>deptBrowse</from-activity-id>
    <control-flow-case>
      <from-outcome>toEdit</from-outcome>
      <to-activity-id>deptEdit</to-activity-id>
    </control-flow-case>
  </control-flow-rule>
  <control-flow-rule>
    <from-activity-id>deptEdit</from-activity-id>
    <control-flow-case>
      <from-outcome>toBrowse</from-outcome>
      <to-activity-id>deptBrowse</to-activity-id>
    </control-flow-case>
  </control-flow-rule>
</task-flow-definition>
```

After you set up a JSF page file, you can drop components such as buttons into the page. The button component's *Action* property can refer directly to the name of the control flow case. For example, the sample application's **Edit Department** button is defined in the JSF page using the following code:

```
<af:commandButton text="Edit Department" id="cb2" action="toEdit"/>
```

When the user clicks this button, the Controller finds the definition of the toEdit action in the task flow file. This code (listed earlier) declares that the flow toEdit defined in the from-outcome tag will load the deptEdit activity (in this case, a JSF page). The **Browse Departments** button on the edit page reverses this navigation using the toBrowse flow.

**Note**: With ADF Controller, as well as with ADF BC, you can always write Java code to supplement or replace functionality. However, the more functionality you can define declaratively, the more you will be using the power of these frameworks.

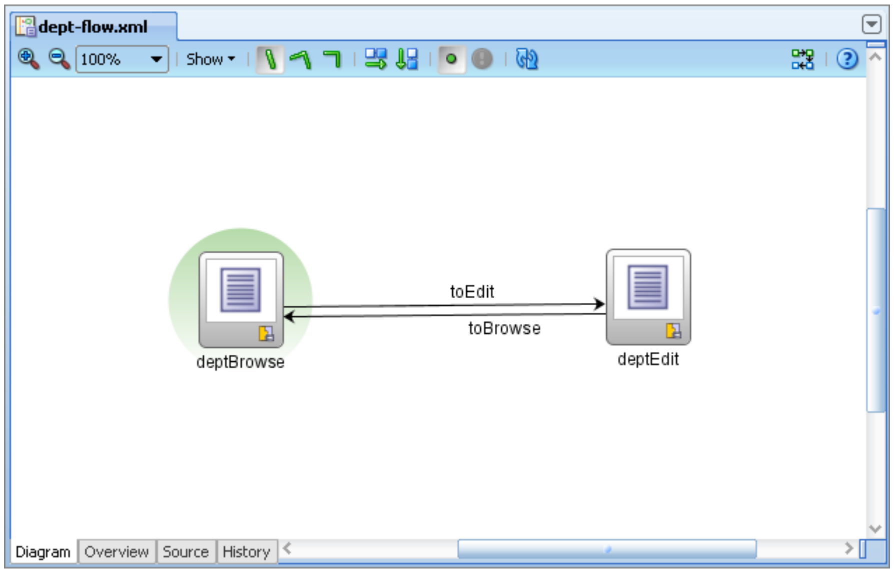


**Figure 4. Task Flow Diagram**

## ADF Faces Rich Client

The ADF View layer constructs the user interface. In the case of a web application, the user interface is rendered in a Hypertext Markup Language (HTML) browser. Native HTML items such as text input items, buttons, selection lists, and radio buttons are limited in functionality. JSF defines higher-level items (called "components") that add functionality to HTML. ADF Faces Rich Client (available in JDeveloper 11*g* and abbreviated hereafter as "ADF Faces") is a set of JSF components with "rich" functionality. For example, ADF Faces offers a component called `af:table` (ADF Faces components are prefixed with "af" denoting the tag library in which they are found) that represents an HTML table in a web browser. Combining `af:table` with one or more `af:column` components allows you to define an entire HTML table without writing HTML. Here is a snippet of code for the Employees read-only table in the sample application:

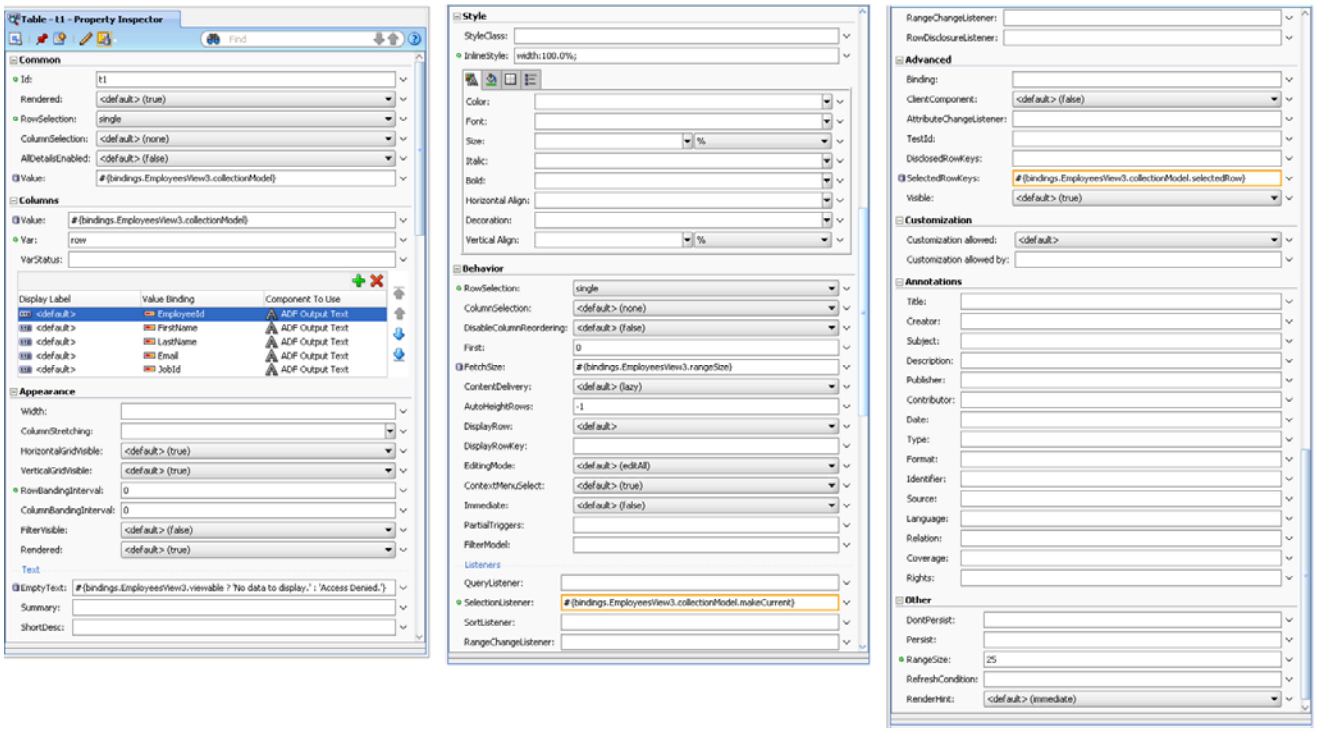
```
<af:table value="#{bindings.EmployeesView3.collectionModel}" var="row"
          rows="#{bindings.EmployeesView3.rangeSize}"
          emptyText="#{bindings.EmployeesView3.viewable ? 'No data to display.' : 'Access Denied.'}"
          fetchSize="#{bindings.EmployeesView3.rangeSize}"
          rowBandingInterval="0"
          selectedRowKeys="#{bindings.EmployeesView3.collectionModel.selectedRow}"
          selectionListener="#{bindings.EmployeesView3.collectionModel.makeCurrent}"
          rowSelection="single" id="t1" inlineStyle="width:100.0%;">
  <af:column sortProperty="EmployeeId" sortable="true"
             headerText="#{bindings.EmployeesView3.hints.EmployeeId.label}"
             id="c2">
    <af:outputText value="#{row.EmployeeId}" id="ot6">
  </af:column>
  <af:column sortProperty="FirstName" sortable="true"
             headerText="#{bindings.EmployeesView3.hints.FirstName.label}"
             id="c5">
    <af:outputText value="#{row.FirstName}" id="ot7"/>
  </af:column>
...
</af:table>
```

Notice that, like the Business Services and Controller layer code, ADF Faces is also XML code consisting of elements ("components" in ADF Faces) and attributes ("properties" in ADF Faces). The power of ADF Faces is in the flexibility of the component properties. In this sample code listing, the *value* property of `af:table` connects the table component to a data source (EmployeesView3 in this case—an instance of the EmployeesView view object) and assigns a variable name (called "row") to each record in the result set of that view object. Nested within the `af:table` component are two `af:column` components—representing the EmployeeId and FirstName attributes. Within each column component is an `af:outputText` (read-only text) component whose *value* property identifies the table data element within a single record (using the variable "row") that will be displayed in the HTML table cell. The `af:table` component is responsible for iterating rows appropriately for the data set.

> **Note**: As discussed more in the next section of this white paper, the “bindings” reference in the af:table component’s value property points to the page binding, which connects the ADF BC objects to the components on the page.

Although the `af:table` code in the preceding snippet is functional code (many properties are defaulted and properties with default values are not represented in code) many more properties are available. Figure 5 shows JDeveloper’s *Property Inspector* (the default property editor for most XML files) displaying the complete set of properties for `af:table.` (This display spreads across three columns although JDeveloper shows all properties in a single column.) You can zoom in for a closer look at individual property names, but the main point is that this component offers a lot of options for modifying its behavior or appearance. Some properties are data-oriented as just explained but some supply user-friendly features such as the following:

- **rowSelection** Setting this property to “single” will allow the user to select a row at runtime (by clicking it). The selected row can then be processed in a way you define (for example, to display a popup showing more detail). You can also define the ability to select multiple rows.
- **rowBandingInterval** Setting this property to “1” will shade every other row in the table to make rows visually easier to follow across a wide display
- **filterVisible** If you set this property to “true,” the table component will display input fields above each column heading. The user can type a value into one or more of these fields and the displayed rows will be filtered by the entered values.



**Common**

| | |
|---|---|
| Id: | t1 |
| Rendered: | <default> (true) |
| RowSelection: | single |
| ColumnSelection: | <default> (none) |
| AllDetailsEnabled: | <default> (false) |
| Value: | #{bindings.EmployeesView3.collectionModel} |

**Columns**

| | |
|---|---|
| Value: | #{bindings.EmployeesView3.collectionModel} |
| Var: | row |
| VarStatus: | |

| Display Label | Value Binding | Component To Use |
|---|---|---|
| <default> | EmployeeId | ADF Output Text |
| <default> | FirstName | ADF Output Text |
| <default> | LastName | ADF Output Text |
| <default> | Email | ADF Output Text |
| <default> | JobId | ADF Output Text |

**Appearance**

| | |
|---|---|
| Width: | |
| ColumnStretching: | |
| HorizontalGridVisible: | <default> (true) |
| VerticalGridVisible: | <default> (true) |
| RowBandingInterval: | 0 |
| ColumnBandingInterval: | 0 |
| FilterVisible: | <default> (false) |
| Rendered: | <default> (true) |

Text

| | |
|---|---|
| EmptyText: | #{bindings.EmployeesView3.viewable ? 'No data to display.' : 'Access Denied.'} |
| Summary: | |
| ShortDesc: | |

**Style**

| | |
|---|---|
| StyleClass: | |
| InlineStyle: | width:100.0%; |
| Color: | |
| Font: | |
| Size: | % |
| Italic: | |
| Bold: | |
| Horizontal Align: | |
| Decoration: | |
| Vertical Align: | % |

**Behavior**

| | |
|---|---|
| RowSelection: | single |
| ColumnSelection: | <default> (none) |
| DisableColumnReordering: | <default> (false) |
| First: | 0 |
| FetchSize: | #{bindings.EmployeesView3.rangeSize} |
| ContentDelivery: | <default> (lazy) |
| AutoHeightRows: | -1 |
| DisplayRow: | <default> |
| DisplayRowKey: | |
| EditingMode: | <default> (editAll) |
| ContextMenuSelect: | <default> (true) |
| Immediate: | <default> (false) |
| PartialTriggers: | |
| FilterModel: | |

Listeners

| | |
|---|---|
| QueryListener: | |
| SelectionListener: | #{bindings.EmployeesView3.collectionModel.makeCurrent} |
| SortListener: | |
| RangeChangeListener: | |
| RowDisclosureListener: | |

**Advanced**

| | |
|---|---|
| Binding: | |
| ClientComponent: | <default> (false) |
| AttributeChangeListener: | |
| TestId: | |
| DisclosedRowKeys: | |
| SelectedRowKeys: | #{bindings.EmployeesView3.collectionModel.selectedRow} |
| Visible: | <default> (true) |

**Customization**

| | |
|---|---|
| Customization allowed: | <default> |
| Customization allowed by: | |

**Annotations**

| | |
|---|---|
| Title: | |
| Creator: | |
| Subject: | |
| Description: | |
| Publisher: | |
| Contributor: | |
| Date: | |
| Type: | |
| Format: | |
| Identifier: | |
| Source: | |
| Language: | |
| Relation: | |
| Coverage: | |
| Rights: | |

**Other**

| | |
|---|---|
| DontPersist: | |
| Persist: | |
| RangeSize: | 25 |
| RefreshCondition: | |
| RenderHint: | <default> (immediate) |

**Figure 5. Property Inspector view of the af:table properties**

## Declarative AJAX

The recent movement to make web applications more interactive has led to acceptance and wide use of *Asynchronous JavaScript and XML (AJAX).* AJAX (sometimes spelled as “Ajax”) consists of a number of technologies that have existed for some time (such as JavaScript and XML); it allows you to write code that refreshes only part of the page instead of the entire page. This enhances the user experience because the user does not need to wait for the entire page to redraw after clicking a button or link, changing a data value, or interacting with the page in some other way.

ADF Faces components are written with embedded AJAX features. For example, in the preceding code listing, the *sortable* property of the **`af:column`** components are set to “true.” This sets up functionality that if the user clicks a column heading the rows displayed will be sorted based on the values in that column. Clicking the same column heading again reverses the sort. As the table is redrawn to display the rows in a different order, the rest of the page stays in place. That is, only the table contents are redrawn. This partial page drawing uses AJAX technology.

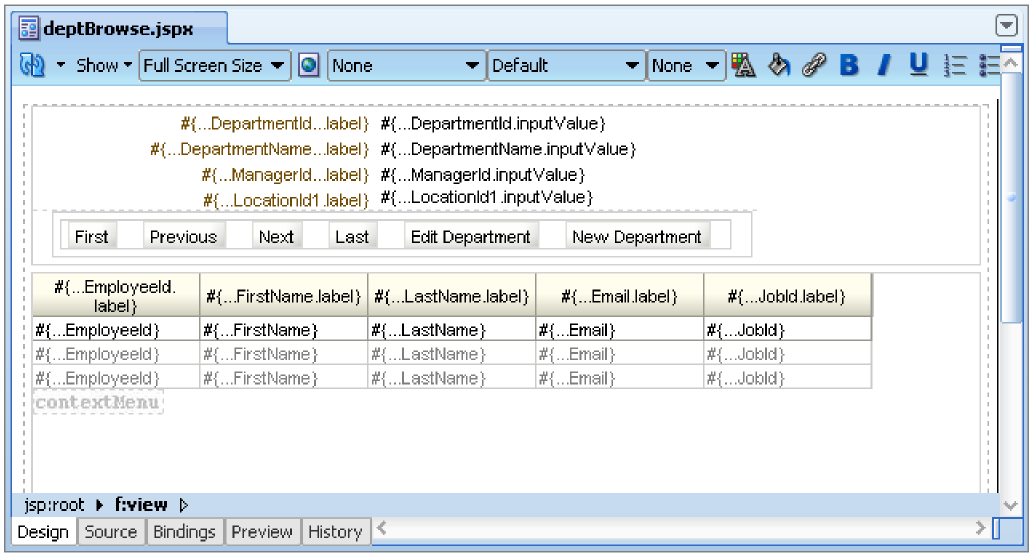
AJAX is built into the ADF Faces components so you do not need to write any JavaScript or XML code to cause the partial page redraw. However, with a small handful of properties, you can write your own partial page events, again without writing AJAX code. For example, by declaring property values for Price, Quantity, and Line Total fields, you can cause a refresh of the Line Total field when the user changes either Price or Quantity fields. The rest of the page would remain static. Only the value in the Line Total field would change when Prince or Quantity changes.

**Note**: AJAX within ADF Faces is more properly called “Partial Page Rendering” (PPR), which specifically refers to the capability to define AJAX functionality by just declaring property values.

## Visual Editor

In addition to the Property Inspector and source code view of the ADF Faces components in JSF file, you can view the components in a visual editor that emulates the component runtime. Figure 6 shows the Departments browse page as it appears in the visual editor.

This tool supports drag-and-drop actions for repositioning components. Changes you make in the visual editor are reflected in the source code just as changes you make in Task Flow Diagram are reflected in the controller source code. As an ADF application developer, you create code in any way that is most efficient and intuitive. For example, it is probably easier to reposition buttons by dragging and dropping them in the visual editor rather than reordering lines of code in the source code editor.

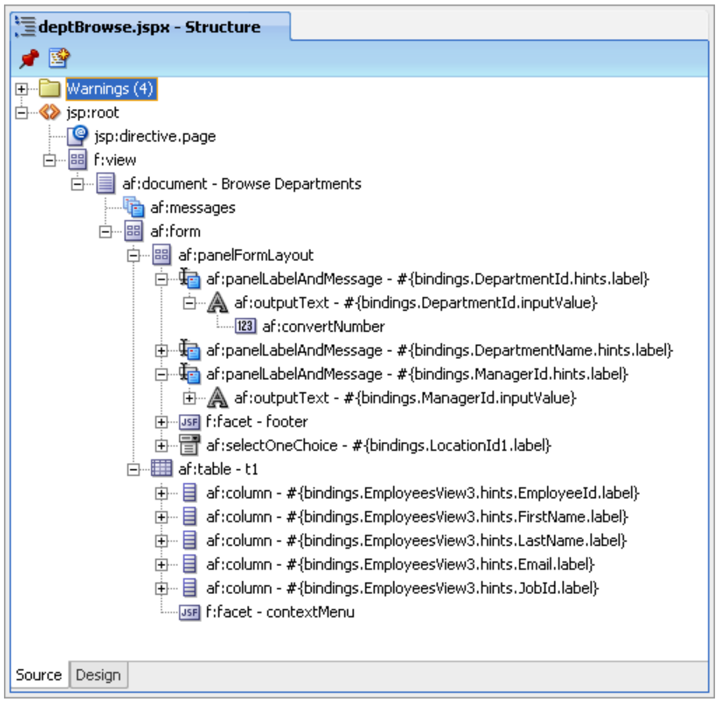


| | |
|---|---|
| #{...DepartmentId...label} | #{...DepartmentId.inputValue} |
| #{...DepartmentName...label} | #{...DepartmentName.inputValue} |
| #{...ManagerId...label} | #{...ManagerId.inputValue} |
| #{...LocationId1.label} | #{...LocationId1.inputValue} |

| #{...EmployeeId.label} | #{...FirstName.label} | #{...LastName.label} | #{...Email.label} | #{...JobId.label} |
|---|---|---|---|---|
| #{...EmployeeId} | #{...FirstName} | #{...LastName} | #{...Email} | #{...JobId} |
| #{...EmployeeId} | #{...FirstName} | #{...LastName} | #{...Email} | #{...JobId} |
| #{...EmployeeId} | #{...FirstName} | #{...LastName} | #{...Email} | #{...JobId} |

**Figure 6. JDeveloper visual editor display of the Departments browse page**

In addition to the visual editor, you can interact with ADF Faces source code (as well as most other types of code) using the Structure window, shown here:



```
deptBrowse.jspx - Structure
Warnings (4)
jsp:root
  jsp:directive.page
  f:view
    af:document - Browse Departments
      af:messages
      af:form
        af:panelFormLayout
          af:panelLabelAndMessage - #{bindings.DepartmentId.hints.label}
            af:outputText - #{bindings.DepartmentId.inputValue}
              af:convertNumber
          af:panelLabelAndMessage - #{bindings.DepartmentName.hints.label}
          af:panelLabelAndMessage - #{bindings.ManagerId.hints.label}
            af:outputText - #{bindings.ManagerId.inputValue}
          f:facet - footer
          af:selectOneChoice - #{bindings.LocationId1.label}
        af:table - t1
          af:column - #{bindings.EmployeesView3.hints.EmployeeId.label}
          af:column - #{bindings.EmployeesView3.hints.FirstName.label}
          af:column - #{bindings.EmployeesView3.hints.LastName.label}
          af:column - #{bindings.EmployeesView3.hints.Email.label}
          af:column - #{bindings.EmployeesView3.hints.JobId.label}
          f:facet - contextMenu
Source  Design
```

This window displays the hierarchy of ADF Faces and JSF component tags and allows repositioning them using drag-and-drop operations. In addition, you can select, delete, and copy nodes in this window to change the source code. The right-click menu on any node allows you to add components above, below, or inside that component. Errors and warnings are summarized at the top of this view and double clicking an error will open the source code editor to the problem line of code.

Although the sample application displays relatively standard interface components, ADF Faces offers nearly 150 components that you can use to create virtually any user interface you can envision. In addition to simple user input items—for example, text items and pulldowns—ADF Faces also supplies more complex input items such as a date input item with calendar popup, a shuttle control that serves as a multiple selection list, and a full-featured calendar widget. It also provides layout components that allow you to manage the relative positioning of components. In addition, a separate set of ADF Faces components called *Data Visualization Tools* (DVT) provides highly-interactive, Web 2.0, Flash-aware components such as graph, chart, gauge, hierarchy viewer, Gantt chart, map, and pivot table.

*Le superflu, chose très nécessaire.*

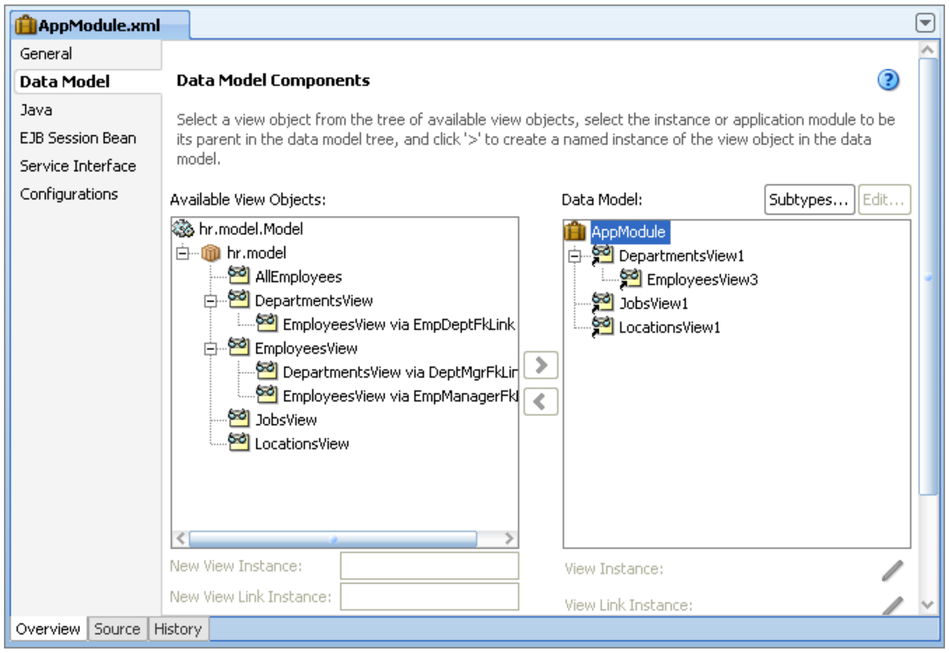
(The superfluous, a very necessary thing.)

—Voltaire *(1694-1778), Le Mondian*

## ADF Bindings and ADF Data Controls

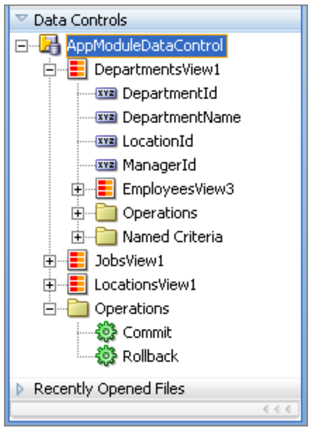
The Model layer in ADF is composed of two aspects—ADF Data Controls and ADF Bindings. These frameworks link the database components written in ADF BC to user interface components (through the management of pages in the Controller layer). Wiring user interface components to database objects is relatively easy with these two technologies.

The story of how these ADF Model layer technologies work starts back in the ADF Business Services layer. An ADF BC component, the *application module*, manages database transactions (COMMIT and ROLLBACK) and defines the *data* *model*, a list of view objects and view links that the application uses. The data model is depicted within the Application Module Editor as a hierarchy as shown here:



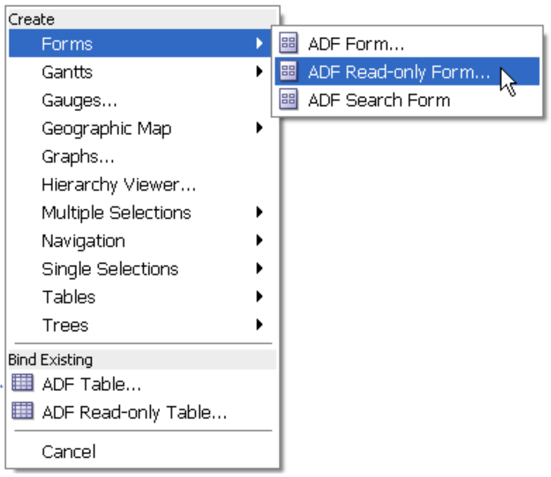
Select a view object from the tree of available view objects, select the instance or application module to be its parent in the data model tree, and click '>' to create a named instance of the view object in the data model.

The *Data Model* area in this example defines view objects for DepartmentsView with a detail of EmployeesView (the suffix numbers indicate distinct usages of the view objects in the data model) linked through a view link. A master-level instance of JobsView and LocationsView (used to supply unfiltered data for pulldowns or LOVs) is also part of this data model. This data model is defined completely within the ADF Business Components application module in the Business Services layer.



Returning to the Model layer, whenever you create a JSF page or page fragment, the Data Controls panel in the JDeveloper navigator will display the ADF BC application module's data model nodes as shown on the right. Additional nodes appear under each view object for attributes (for example, DepartmentId under DepartmentsView1), Operations (that provide actions you can take on the data collection, such as navigating the current record to the Next, Previous, First, or Last record in the set), and Named Criteria (which define which fields will be available for queries using search forms).

An almost magical thing occurs when you drag one of these nodes onto a JSF page or page fragment. For example, to build the sample application, the DepartmentsView1 node was dragged from the Data Controls panel and dropped onto the JSF page. The ADF Data Controls framework determines that the node is a collection-level (table-level) item and displays a menu of applicable components or component combinations as shown on the left with the **Forms** menu expanded.



In the sample application, selecting **ADF Read-only Form** caused JDeveloper to create a display containing labeled fields with navigation buttons at the top of the Departments browse page. This drag-and-drop-and-select action interacts with the ADF Data Controls list. If an individual attribute node (such as DepartmentId) is dragged instead, a list of data controls appropriate to a single data value (for example, input text items, output items, and pulldowns) will display instead.

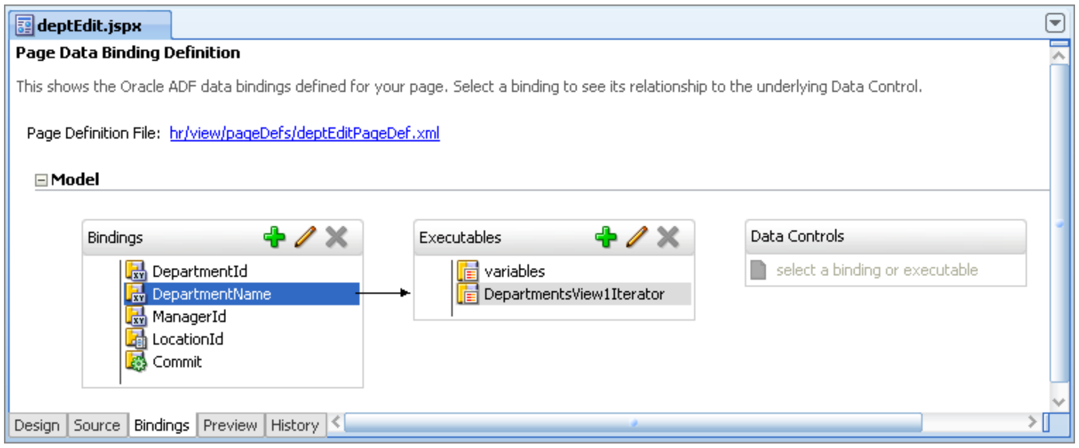
In addition to drawing user interface components on the screen, the drag-and-drop operation also creates bindings for those components.

*Bindings* are code or definitions that declare which data from a business service will be connected to a user interface control or structure. Bindings appear in the ADF Faces' property values. The following example is an ADF Faces input text component from the Edit Department page:

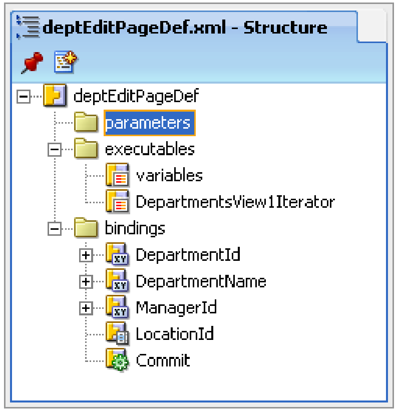
```
<af:inputText value="#{bindings.DepartmentId.inputValue}"
            label="#{bindings.DepartmentId.hints.label}"
            required="#{bindings.DepartmentId.hints.mandatory}"
            columns="#{bindings.DepartmentId.hints.displayWidth}"
            maximumLength="#{bindings.DepartmentId.hints.precision}"
            shortDesc="#{bindings.DepartmentId.hints.tooltip}"
            id="it1">
</af:inputText>
```

All of this code was created by the Data Controls panel drag-and-drop operation. This is one of the main advantages of the Data Controls panel: it builds all the property values for you and automatically binds the components to data. The property values defined using the "#{ }" delimiters are Expression Language expressions. *Expression Language* (EL) is a high-level, non-procedural language specified in the JavaServer Pages standards. It is used within JSF pages to refer to potentially dynamic sources of data that will supply property values at runtime.

In this case, all EL expressions begin with "bindings," which is the context for the values. This context refers to a PageDef (Page Definition bindings) file that JDeveloper creates for each JSF page. You can view the bindings in this file using the Bindings viewer for the page as shown here:


deptEdit.jspx
Page Data Binding Definition
This shows the Oracle ADF data bindings defined for your page. Select a binding to see its relationship to the underlying Data Control.
Page Definition File: hr/view/pageDefs/deptEditPageDef.xml
Model
Bindings
DepartmentId
DepartmentName
ManagerId
LocationId
Commit
Executables
variables
DepartmentsView1Iterator
Data Controls
select a binding or executable
Design
Source
Bindings
Preview
History

If you need to look at or manipulate the bindings code, you click the link next to the Page Definition File label to open the PageDef file—the container for the bindings definitions. The Structure window view of this page is shown on the right.



You will see an *executables* section for the queries (iterators) that occur when the page opens. You will also see a *bindings* section for the objects that refer to view object attributes. By now, you will not be surprised that JDeveloper creates XML code to define bindings; you will rarely need to touch this code. Here is a code snippet from the deptEditPageDef.xml file:

```
<bindings>
  <attributeValues IterBinding="DepartmentsView1Iterator"
      id="DepartmentId">
    <AttrNames>
      <Item Value="DepartmentId"/>
    </AttrNames>
  </attributeValues>
```

This file is processed by the ADF Bindings framework code and links the attribute, DepartmentId, to the iterator, DepartmentsView1Iterator. That iterator is defined for the DepartmentsView1 view object instance in the data model, and therefore represents a query of data. The EL bindings expressions in the ADF Faces component code point to this communication path and therefore to data. The EL expressions also further drill into a specific property of the ADF BC view attribute; for example, the label property of the example component is defined as "#{bindings.DepartmentId.hints.label}," which refers to the label property of the view attribute (in the hints property category). If no label property is defined, the default label is the attribute name.

## Which Languages Are Important?

Now that you have sampled some ADF techniques for working with each of the core technologies, you know that JDeveloper creates a lot of application code automatically when you interact with its visual and declarative tools. However, you may still be wondering about which languages you will use when you need to supplement this code. First, remember that ADF was created as a visual and declarative environment to interact with many frameworks. Therefore, a key skill is knowing how to squeeze the most functionality out of the technologies by just defining property values and laying out components visually. The less code you need to write, the less code you need to debug.

With the goal of "declarative if at all possible" in mind, you can be quite productive without writing much code. However, you will come to a point where writing code is necessary and you will be using a combination of languages. The following list summarizes the main languages you need to know and how you will use them:

- **XML** As you have seen, work with frameworks makes heavy use of XML code. However, you work with most XML code in JDeveloper using declarative and visual tools. You will rarely need to type XML elements and attributes in these files, but the level of skill you will need at that time is very basic. You mainly need to know three things about XML: elements need ending tags; elements have attributes that refine the element's use; and elements can be nested within elements to create an element hierarchy.
- **Java** You will write snippets of Java inside ADF BC classes and View layer code to perform customized tasks that the frameworks cannot provide. You can be quite productive in the ADF Fusion Technology Stack with a novice level knowledge of Java if you have someone on your team who understands Java at an expert level. This person can step in to assist if you run into a requirement that cannot be handled with a basic knowledge of Java.
- **HTML** For best use of JSF and ADF Faces concepts, you will avoid writing HTML code. Instead, you use high-level components that generate HTML for you.
- **Cascading Style Sheets (CSS)** ADF Faces components use CSS styles defined in a *skin*, a set of style selectors that provide a common look-and-feel to all your pages. You will use CSS to define the skin at the start of the first ADF application project, but will not need it much after that because you will apply the same skin to all applications in your organization.
- **JavaScript** ADF Faces components use JavaScript internally to provide user-friendly features such as refreshing part of a page when scrolling to the next set of records. You will usually not need to write JavaScript or AJAX code because the components provide many of the features you would normally need other languages to supply and allow you to declare AJAX functionality using only property values.
- **Expression Language** EL is used to supply dynamic values to JSF components' properties. The main learning curve for EL is in knowing how to start to build the correct expression. Fortunately, JDeveloper can assist. In the pulldown for most properties is an item for "Expression Builder." This selection displays a navigator that helps you create properly formatted EL expressions. It is a good learning tool as well as a way to enter proper EL.
- **Groovy** ADF Business Components allow you to write validation and message code using this language. As with EL, Groovy is used at a very basic level and understanding a few fundamentals as explained in the JDeveloper online help system will suffice.

## Additional Resources

The intention of this white paper is to get you started thinking about ADF, Fusion, and techniques you will be using in JDeveloper to build web applications. The main source of all things ADF is the JDeveloper home page on Oracle Technology Network (www.oracle.com/technology/products/jdev/). Follow the links on that page to access tutorials and articles about specific techniques. In addition, the "Learn More" tab on that page currently displays a "Learn More About" link to information about ADF. The Technical Resources section contains links for Developer Guides; the "Fusion Developer's Guide" is a good starting point for a wealth of technical detail about ADF. This Oracle website guide is linked to within the JDeveloper help system as well.

Speaking of the help system (technically called the "Help Center"), the Help menu contains link to Tutorials (v.11.1.2) or Cue Cards (11.1.1), which step you through creating a specific type of code and are especially helpful when learning ADF. Another extremely useful Oracle resource can help in learning about ADF Faces RC components: the ADF Faces Rich Client Components Hosted Demo (available at jdevadf.oracle.com/). This demo shows all ADF Faces components and allows you to change properties to see how they work.

*Le secret d'ennuyer est celui de tout dire.*

(The secret of being a bore is to tell everything.)

—Voltaire (1694-1778), *Sept Discours en Vers sur l'Homme*

## CONCLUSION

Admittedly, this is a lot of information but, hopefully, you now have a better idea about Oracle Fusion and ADF as well as about the basics about each of the core technologies in the ADF Fusion Technology Stack: ADF BC, ADF Controller, ADF Faces RC, and ADF Model: Bindings and Data Controls. This white paper has shown the type of code you will be creating and the style of development work you will be performing to create that code in each of these technologies. This overview information should help in your understanding of what you need to know to be productive with ADF and Fusion technologies and to start up the on-ramp to the Fusion Development Highway.

May that road rise up to meet you!

*Il faut cultiver notre jardin.*

(Let us cultivate our garden.)

—Voltaire (1694-1778), *Candide* (Ch. xx)

## About the Author

**Peter Koletzke** is a technical director and principal instructor for the Enterprise e-Commerce Solutions practice at Quovera, in Palo Alto, California, and has over 30 years of industry experience. Peter has presented at various Oracle users group conferences around 320 times and has won awards such as Pinnacle Publishing's Technical Achievement, Oracle Development Tools Users Group (ODTUG) Editor's Choice (three times), ODTUG Best Speaker, ODTUG Volunteer of the Year, NYOUG Editor's Choice (three times), and ECO/SEOUC Oracle Designer Award. He is an Oracle Certified Master, Oracle ACE Director, and coauthor (variously with Dr. Paul Dorsey, Avrom Roy-Faderman, and Duncan Mills) of eight Oracle Press development tools books including *Oracle JDeveloper 11g Handbook*.