Oracle JDeveloper 10g

Reviewers Guide

March 2004

ORACLE

INTRODUCTION

This guide will help you experience application development with Oracle JDeveloper 10*g*. This guide provides step-by-step instructions that leads you through using various features and technologies in Oracle JDeveloper.

This guide, however, doesn't cover the full breadth of functionality available in Oracle JDeveloper 10g. It focuses on the process of building J2EE applications using various technology stacks. It doesn't cover other JDeveloper features such as UML modeling, code debugging, application profiling, code tuning, Swing and J2ME clients, and database modeling.

For more information, tutorials, demos, and online help, visit the Oracle Technology Network (OTN), at http://otn.oracle.com/products/jdev/.

ORACLE JDEVELOPER 10*g* OVERVIEW

Oracle JDeveloper 10*g* is an integrated development environment (IDE) for building applications and Web services using the latest industry standards for Java, XML, and SQL.

Oracle JDeveloper supports the complete development life cycle with integrated features for modeling, coding, debugging, testing, profiling, tuning, and deploying applications.

A visual and declarative development approach and the innovative Oracle Application Development Framework (Oracle ADF) work together to simplify application development and reduce mundane coding tasks, offering unparalleled productivity and a choice of technology stacks.

Complete and Integrated

Oracle JDeveloper supports every step of the development life cycle, including modeling, coding, debugging, testing, profiling, tuning, and deploying applications. All these tasks are done from a single IDE using a set of integrated features.

Oracle JDeveloper focuses on Java application development using J2EE, J2SE, or J2ME. In addition, JDeveloper enables XML-based application development with features such as the XML Schema Modeler, XML code insight, and the XML tag property inspector. To complete the developer's toolbox, Oracle JDeveloper also provides a full development and modeling environment for building database objects and stored procedures.

Oracle JDeveloper provides a single, highly integrated, developer-friendly IDE, with a consistent interface and development experience.

Productivity with Choice

The goal of Oracle JDeveloper 10g is to make J2EE development simpler and more accessible. To achieve this goal, Oracle JDeveloper focuses on a visual and declarative approach to J2EE development. Further simplification is provided by the Oracle Application Development Framework (Oracle ADF) - a J2EE development framework that implements design patterns and eliminates infrastructure coding.

A unique aspect of Oracle JDeveloper is that the same productive development experience is used for various technology stacks. For example, developers can choose to implement a persistence layer using simple Java classes, EJB, TopLink, Oracle ADF Business Components, or Web services. Regardless of the chosen technology, Oracle JDeveloper will provide a declarative way to create this layer, as well as drag-and-drop mechanisms to bind user interface components to any of these implementations.

Realizing that different developers have a different Java skill level and their own preferred development approach, Oracle JDeveloper provides a choice of development approaches that includes Model Driven Architecture (MDA), declarative development, and hand-coding. Developers can choose the approach that best suits their personal development style.

Applications developed with JDeveloper work with any data source and can be deployed on any J2EE-compatible application server.

Oracle JDeveloper, a 100% Java-based tool, is a cross-platform IDE that runs on Windows, Linux, and various Unix-based systems, providing a choice of development platform.

Standard, Open, and Extensible

Oracle JDeveloper enables developers to use the latest industry standards to develop applications that can operate across multiple hardware and software platforms. Applications built with Oracle JDeveloper can be deployed to any J2EE-compliant server and can access any JDBC-compliant database.

Oracle JDeveloper embraces popular open source frameworks and tools, providing built-in features for Struts, Ant, JUnit, and CVS. This integration

enables developers to use these open source tools to streamline their development process.

Oracle JDeveloper offers an Extension SDK that lets developers add capabilities and customize the development environment. Oracle JDeveloper is built as a set of extensions on top of a core IDE platform. Developers can turn extensions on or off as they wish, customizing the IDE for their needs. The same API that is used by the JDeveloper team to develop the product is available to developers and third party companies who are interested in integrating with and enhancing Oracle JDeveloper.

J2EE DEVELOPMENT AND ORACLE ADF

This reviewer's guide focuses on J2EE development using Oracle JDeveloper 10g.

J2EE is a set of specifications for building multi tier applications using the Java language. J2EE is a robust, scalable, and secure platform that forms the basis for many of today's enterprise applications.

Over the years, best practices and design patterns have evolved for the J2EE platform. The problem is that implementing these best practices usually involves writing a lot of infrastructure code. Oracle JDeveloper 10g aims to solve this problem.

Oracle JDeveloper 10g includes the Oracle Application Development Framework (Oracle ADF). This framework simplifies J2EE development by minimizing the need to write code that implements design patterns and application's infrastructure. Oracle ADF provides them as part of the framework. Oracle ADF features both runtime services and development features.

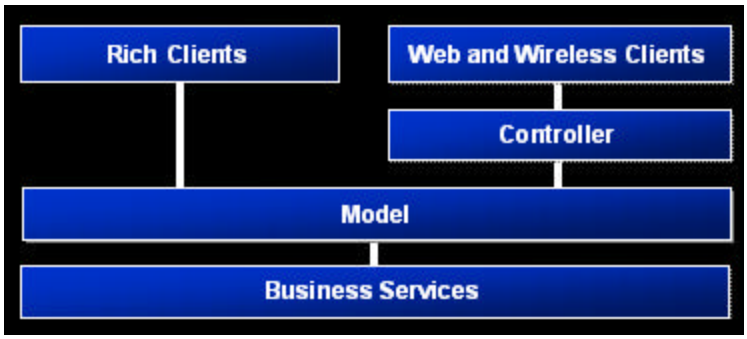
Oracle ADF is an evolution, an improvement, and an extension of frameworks that were included with previous versions of JDeveloper.

Oracle ADF is based on the Model-View-Controller (MVC) design pattern. MVC separates the application architecture into three layers:

- Model - handles interaction with data-sources and runs the business logic
- View - handles the application user interface
- Controller - handles the application flow and acts as the interface between the Model and the View layers

Separating applications into these three layers simplifies maintaining and reusing components across applications. The independence of each layer from the others results in a loosely coupled architecture. Oracle ADF provides an easy way to implement the MVC architecture.

The Oracle ADF Layers



Oracle ADF – high level architecture

Oracle ADF is based on four layers:

- The Business Services layer - provides access to data from various sources.
- The Model layer - provides an abstraction layer on top of the Business Services layer, enabling the View and Controller layers to work with different implementations of Business Services in a consistent way.
- The Controller layer - provides a mechanism to control the flow of the Web application.
- The View layer - provides the interface to the application.

Oracle ADF lets developers choose the technology they prefer to use when implementing each of the layers. Oracle ADF provides the same visual and declarative development experience regardless of the technology stack used.

The View Layer

The View layer provides the user interface to the application. The view layer uses HTML, rich Java components, or XML and its variations to render the user interface. The View layer can be Web-based, client server-based, or even a wireless implementation.

The Controller Layer

The Controller layer controls the application's flow. Web-based applications are composed of multiple Web pages with dynamic content. The controller layer manages the flow between these pages. Different models can be used when building this layer. The most prominent architecture for Java-based Web applications relies on a servlet that acts as the controller. The Apache Jakarta Struts controller, an open source framework controller, is the de facto standard

controller for Java-based Web systems. Oracle ADF uses the Struts controller to manage the flow of Web applications.

The Business Services Layer

The Business Services layer manages interaction with a data persistence layer. It provides such services as data persistence, object/relational mapping, transaction management, and business logic execution.

The Business Services layer in Oracle ADF can be implemented as simple Java classes, EJB, Web services, TopLink objects, or Oracle ADF Business Components.

The Model Layer

The Model layer connects the Business Services to the objects that use them in the other layers. Oracle ADF provides a Model layer implementation that sits on top of Business Services, providing a single interface that can be used to access any type of Business Service. Developers get the same development experience when binding any type of Business Service layer implementation to the View and Controller layers. The Model layer in Oracle ADF served as the basis for JSR-227 "A Standard Data Binding & Data Access Facility for J2EE". For more information about this JSR, visit http://web1.jcp.org/en/jsr/detail?id=227.

GETTING TO KNOW THE IDE

In the following tutorial you will get to know the different components that make up the JDeveloper interface. You will also learn about different ways you can interact with the code you develop.

Technology Scopes

One way Oracle JDeveloper simplifies the Java world to developers is by using Technology Scopes. Java has many APIs and technologies that can be used when building an application. However, each project usually uses only a specific set of these technologies. In Oracle JDeveloper you can define applications templates that contain a certain set of technologies that will be used in this specific application. This is the "Technology Scope" for that application. Defining a specific technology scope means that JDeveloper will only show the relevant options in the menus and galleries for this specific project. JDeveloper comes pre-packaged with several predefined technology scopes and you can add your own technology scopes.

Let's define a new Technology Scope in JDeveloper.

1. Start up Oracle JDeveloper 10*g* (run the file [jdevdirectory]\jdev\bin\jdevw.exe).
2. In the menu choose **File->New** and in the **General** node choose **Application Workspace**.
3. Have a look at the pre-defined **Application Templates** looking at their description.
4. Click on the **Manage Templates** button
5. Click the **New** Button to create a new Application Template
6. Give a name to your template
7. While standing on the your new template click the **New** button again to create a project template
8. Specify the project template and project name in the dialog and click ok.
9. In the available technologies list shuttle **Java** and **Extensible Markup Language (XML)** to the right. Click **OK.**

10. Complete the creation of your application by filling the values in the **create application workspace** dialog making sure you are using the application template you just defined.

You are now ready to develop your first application.

Code Editing Features

In this section you'll see how Oracle JDeveloper 10*g* provides you with a helpful code editor as well as utilities that can save you mundane coding.

First let's see the ways the code editor helps you write code faster.

1. Right Click on the new project you created and choose **New**.
2. Note how the gallery only shows you options related to your specific Technology Scope. Change the **Filter By** list to show **All Technologies** to see the full range of available technologies, and then change it back to your project technologies
3. Choose **Java Class** from the **General** menu.
4. Check the 'Create Main Method' checkbox and click OK.
5. The source code of your new class is opened in the code editor.
6. Enter the following code in the main method **URL u = new URL("http://otn.oracle.com");**
7. The blue line beneath URL indicate that this type of object is not known. Place your cursor on URL and note the tool tip that suggests the correct import that will fix the problem.
8. Press **Alt+Enter** to import java.net.URL.
9. There is another curly blue line under **new URL("http://otn.oracle.com")** this indicates that there is a semantic problem with this section.
10. Place the mouse cursor above this section to see a tool tip that explains the problem. In this case it is a missing exception that we didn't catch.
11. To define a try-catch block easily, click and highlight the line problematic line and from the right click context menu option select **surround with**.
12. From the list of options choose **try-catch**. Note that the correct exception is automatically caught.
13. Your cursor is now position in the **catch{}** section. We want to add code here to print the error message. Oracle JDeveloper allows you to use code templates to speed up your coding. Type **sop** and press ctrl+enter and this will expend to **System.out.println();** statement.

14. You can see other code templates add your own templates and define many of the properties of the code editor in the **tools->preferences->Code Editor** menu option.

15. Getting help is also very simple in Oracle JDeveloper 10g. For example highlight **URL** in your code and press **ctrl+d,** to pop-up online help for the URL object.

As you continue coding you'll encounter other helpful features like code insight into function, syntax error highlighting and more.

Visual Development

In this section we'll see how Oracle JDeveloper 10g can minimize manual coding by using a visual and declarative approach instead.

1. Right Click on the new project you created and choose **New**.
2. Choose **JavaBeans->Bean** from the gallery.
3. Accept the defaults in the **Create Bean** dialog.
4. Your new bean is opened in the Visual Editor.
5. Make sure that the Property Inspector window is open (ctrl+shift+I).
6. Change the l**ayout** property of the panel to **XYLayout.**
7. Make sure the component palette is open (**ctrl+shift+p**) and switch it to show the **Swing** components.
8. Click on the **JTextField** object and place it on the panel. You can drag and drop it to change its location and size.
9. In the **property inspector** change the **background** property of the button to another color.
10. Click on the **Source** tab at the bottom of the visual editor to switch your view of the bean to show the actual Java code. Notice the lines of code that were generated from your visual manipulations.
11. To see both the visual and code editor at the same time use the **splitter** at the top of the scroll bar to split the screen horizontally. So you can see the **Design** editor on top and **Source** editor on the bottom. (another way to split the screen is to **right click** the **tab** for the file you are editing and choose **Split Document**).
12. Change the values of the JTextField background color in code and see it change in the design editor and vice versa.
13. If you want to see multiple files at the same time you can **drag the top ta**b with the name of the file either **to the bottom of the screen** or to the side to further split your code view.

14. Clicking on the **Class tab** in the editor will give you an easy wizard interface for adding Fields, Methods and Events to your bean. Add a field and a method to your bean.
15. When your code gets longer the **Structure window** can help you navigate through it faster.
16. Click around the structure window to see how you are directed to the right place in the code. Explore the various view options of the structure window through the icons at the top. The tooltips for each icon will explain what they are doing.
17. From the **File** menu choose **New->General->Diagrams->Java Class Diagram**.
18. Accept the defaults in the dialog that pops up.
19. From the application navigator **drag and drop** the **files** you've been working on **onto the diagram**.
20. The UML Class Diagram shows you another visual way to work with your classes. Try changing methods and attributes in the diagram and watch them change in the code and vide versa.

Oracle JDeveloper offers this type of synchronization between code and visual representation for both Swing and JSP/HTML clients. You will get to work with the JSP/HTML visual editor in the J2EE tutorials that follow.

Developing XML Applications

Beyond Java coding, Oracle JDeveloper 10g also provides helpful features for the development of XML based applications. In this section we'll step through a couple of the XML related features

1. From the menu choose **File->Open**.
2. Choose open the following file: [JDeveloper install directort]\jlib\schemas.zip\xsd\XMLSchema.xsd
3. This is a XML Schema document. Oracle JDeveloper provides a visual repersentation of the file for easier development.
4. Step through the diagram clicking to expend different nodes.
5. Note that when you click on a node or an attribute you can access their properties in the **property inspector**.
6. You can **split the editor** to view both the Design view and the actual XML source.
7. The **structure pane** on can help you navigate the XML file easily.
8. Right click on the project and choose **New->XML->XML Schema**

9. Click on the exampleElement in the editor and from the component palette choose attribute to add an attribute to your new XML Schema element. Rename the attribute to a meaningful name.
10. Switch to the Source view.
11. Start a new empty line beneath the attribute line and write **<**
12. If you'll wait a second the editor will help you through the coding task by offering the next step **xsd:**
13. As you continue choosing options the editor helps you by offering the possible options at each stage.

    As you can see the visual and declarative approach to coding is available to XML development as well as for Java development.

J2EE DEVELOPMENT TUTORIALS

This section contains tutorials that will guide you through the steps of creating J2EEE applications with various technology stacks. It is recommended that you follow the guide flow to get the maximum understanding of the different technologies and their benefits, but you can also jump to specific section and develop each on its own.

While you are developing the application notice how little Java coding you are doing manually and how much easier the development process is with the visual and declarative approach.

At each step feel free to switch to the source view of the object you are working on, in order to see the actual Java and XML code behind the scene.

SIMPLE STRUTS APPLICATION

In this section we'll create a simple login procedure for the application; using basic JSP combined with a simple Struts page flow.

In the following section you'll get to know the basic concepts of Struts, and you'll see how it separates your view layer from your controller.

The application is a simple password verification process. Once we are done with this part will continue with a more complex application that interact with a database or Web services backend.

Creating a New Application

We'll start by creating a new application.

1. From the **File** menu choose **New**.
2. In the **General** section choose **Application Workspace.**
3. Update the **Application Name** to **loginApplication.**
4. Update the **Application Package Prefix** to **loginapp.**
5. Choose **Web Application [Default]** as the **Application Template**.
6. Click **OK**.

A new application is created in the Applications Navigator. You will notice that the application is divided into a model and a view/controller section. This will help you separate the different layers.

Designing the page flow diagram

In this section we'll design the flow of our application using the visual editor for the struts-config.xml file. The page flow modeler provides a visual representation of the content of this XML file.

1. Right Click on the **ViewController** node in the application navigator and choose **Open Struts Page Flow Diagram**.
2. From the **component palette** choose **Page** and click in an empty space in the diagram. Rename the page to **/login.jsp**.
3. Click on **Action** in the component palette and click in an open space under the /login.jsp icon in the diagram. Rename the action to **/login**.
4. From the component palette choose **Page** and create a new page in the diagram renaming it to **/menu.jsp**.
5. From the component palette choose **Forward** and draw a link by clicking on the **/login** action and then on the **/menu.jsp** page. This is the flow in case the validation succeeds.
6. Create another **Page** in the diagram and rename it to **/failure.jsp** .
7. Create another **Forward** link starting from the **/login** action and going to **/failure.jsp. Rename** this forward link **fail**.

This completes the flow of the login process.

Editing JSP in the Visual Editor

From the page flow modeler you can directly access the source of the pages to edit them. In this section we'll use the HTML/JSP Visual Editor to design our pages.

1. We'll edit your login page using the visual HTML editor. **Double click** on the **/login.jsp** icon to open the page in the visual editor.
2. Write some text for your page heading like "Welcome to the HR application" and in a line below that write "Please login to the application". **Mark the first line** and using the formatting tools at the top of the editor change its type to **heading1**. Mark the second line and change it to **heading2**.
3. In the **component palette** change the view to see **CSS** instead of HTML objects. Click on the **JDeveloper** CSS to assign this CSS to the page.
4. Change the **component palette** view to see **Struts HTML** components.
5. Place the cursor under the text and from the component palette choose **form.**
6. In the dialog that pops up set the **action** property to **login.do** and click **ok**.
7. Change the **component palette** view to see **HTML** components.
8. While the cursor is still placed inside the Struts Form, choose the Table component to create a 2 by 2 table.

9. In the **top left** cell of the table write **Password**.
10. Place the cursor in the **top right cell** of the table and choose **Password Field** from the HTML component palette.
11. In the **bottom right cell** insert a **Submit Button** from the HTML component.
12. Click on the password field and in the property palette change its **name property** to **password.**
13. This completes the login form layout setting. Now that the View layer is finished lets set the correct information for the controller layer. In the next steps will work with the Struts framework to add a processing of the login procedure.

Editing the other pages.

1. Switch back to the Struts-config.xml tab to see the page flow diagram.
2. Right click on an empty space in the diagram and choose D**iagram->Refresh Diagram from All Pages**. This will add a page forward link to your diagram based on the action setting you set in the login.jsp page.
3. Double click on the **/menu.jsp** page to edit it and **write "Welcome to the application".**
4. Switch back to the page flow diagram.
5. Double click on the **/failure.jsp** page to edit it and **write "login failed , please try again".**
6. Switch back to the page flow diagram.
7. Create a **Page Link** starting from **/failure.jsp** going back to **/login.jsp**.

Creating a Struts Action

In this section we'll add the business logic to our Controller layer that will define the application flow. We'll start by adding a Struts formbean, a formbean is the way information is passed between the controller and the view layer.

1. Make sure you can see the Structure window **ctrl+shift+s.**
2. **Right click** on the **Struts Config** node and choose **New -> Forms Beans.**
3. **Right click** on the newly created **Form Beans** node and choose **New -> Form Bean**.
4. While standing on the newly created Form Bean in the **Property Inspector** change the **name** property to **login**.
5. Click on the **type** property and the browse button and navigate to choose **org.apache.struts.action.DynaActionForm** .

6. In the structure window **right click** the **login** Form Bean and choose **New->Form Property**.
7. Change the **name** property of the newly created property to **password**.

The next steps will associate the newly create Form Bean with the Action that use it. It will also show you another way to interact with the struts-config.xml file.

2. In the application navigator expand the tree to locate the **struts-config.xml file** under ViewController->Web Contetnt -> WEB-INF. Right click on the file and choose **Edit Struts Config**.
3. Go to the Action Mapping node and click the login action.
4. Switch to the **Form Bean tab.**
5. From the **name list** choose **login**.
6. Click OK to exit.

Now let's add the actual logic of the login action.

1. Back in the struts-config.xml diagram **double click** the **/login** action icon to create the action. Accept the defaults in the dialog and click **ok**.
2. The code editor is now open with a new method called execute in it. We'll add the logic to this method.
3. **Replace** the line **return mapping.findForward("success");** with the following code.

```
String password = (String)((DynaActionForm)form).get("password");
if (password.equalsIgnoreCase("tiger"))
 {return mapping.findForward("success");}
else
 {return mapping.findForward("fail");}
```

4. Use the import assistant (Alt+Enter) to import the missing org.apache.struts.action.DynaActionForm;
5. **Right click** somewhere in the source code and choose **make** to compile your code.
6. Now we are ready to run our login procedure. Back in the struts-config.xml diagram **right click** on **/login.jsp** and choose **run**.

Try logging in with different passwords (which are hidden as you type) and notice that only "tiger" gets you into the main menu.

Note that you only needed to do Java coding to implement the actual business logic that is used to validate the password. This is the benefit of Oracle ADF – it takes care of the plumbing for you and let you focus on writing the business logic.

ORACLE ADF BUSINESS COMPONENTS APPLICATION

Oracle ADF Business Components is an evolution of the Business Components for Java (BC4J) framework. ADF Business Components aim to simplify building database driven applications by providing a complete framework develop for this specific task. It provides an easy way to map Java objects to tables, automatically managing persistence, O/R mapping, transaction management and resource pooling. ADF Business Components provide an easy way to define validation rules as well as presentation layer preferences for attributes. Oracle ADF Business Components can be deployed as stand-alone Java classes or as a Session EJB.

In the next section you'll build a simple database driven application that allows users to browse and edit details for Departments and Employees.

Creating the Business Services and Model Layers

Creating a Connection to the Database

Oracle JDeveloper lets you browse and develop database objects. To do this you'll need to set-up a connection to your database.

(We are using the HR sample user that you get with a default installation of the Oracle Database. You might need to unlock the user after the default installation using the following command:

ALTER USER "HR" IDENTIFIED BY "hr" ACCOUNT UNLOCK;
If you are encountering any problems with this step you might want to ask your DBA for help).

1. Switch to the **Connections Tab**.
2. Right click the **Database** and choose **New Database Connection . . .**.
3. In the Database Connection Wizard, review the information on the Welcome page and click **Next**.
4. In the Connection name field type **HR**.
5. Click **Next**.
6. On the Authentication page:
    a. In the Username and Password fields, type **HR**.
    b. Select **Deploy Password**.
    c. Click **Next**.
7. On the Connection page:
    a. In the Host name field, type the name (or IP address) of the computer where the database is located. The default 'localhost' should work if the database is installed on your local PC.

b. In the SID and Port fields, type the information for this connection. If you do not know these values, check with your database administrator. (The default 'SID' is 'ORCL' and the default port is '1521')

c. Click **Next**.

8. Click Test Connection. If the database is available and the connection details are correct, you will see the message "Success!". If an error occurs, verify the settings with your database administrator.

9. Click **Finish**.

Creating a New Application

We'll start by creating a new application.

7. From the **File** menu choose **New**.

8. In the **General** section choose **Application Workspace.**

9. Update the **Application Name** to **Hrapplication.**

10. Update the **Application Package Prefix** to **Hrapp.**

11. Choose **Web Application [Default]** as the **Application Template**.

12. Click **OK**.

Modeling Oracle ADF Business Components

For the Business Services layer we want to have Java objects that can interact with the database. We'll use Oracle ADF Business Components for this layer. Oracle ADF Business Components (an evolution of the BC4J framework from previous JDeveloper versions) is a framework that manages every aspect of interaction with your database, including tasks such as Object Relational Mapping, Persistence, Transaction Management and Connection Pooling.

You can create Oracle ADF Business Components directly from a wizard, but will go through a UML modeler to get a better picture of what we are doing.

2. In the **Model** node right click and choose **New**.

3. Choose **Diagrams** and **Business Components Diagram** from the gallery.

4. Rename your diagram and Click **OK**.

5. Make sure that you can see the **connection Tab** and expand the **HR** node so you'll be able to see the various **Tables** in this schema.

6. **Drag and drop** the **Departments** and **Employees** table onto the diagram area.

7. The dialog window lets you choose what to create from the tables. If you switch to see the All technologies instead of just the Project Technologies

you'll see that you can create Enterprise JavaBeans or simple Java Objects from the tables in the same way. But for this project choose the **Business Components Entity Objects** and click **OK**.

The diagram shows you the two objects, their attributes and the relationships between them**.**

Adding Simple Validation

Oracle ADF Business Components provide a very easy way to add simple validation rules to objects. Lets add a validation that will make sure the value of an employee salary is inside a specific range.

7. **Double click** on the **Employee** Entity Object in the diagram to open the Entity Object Editor.
8. Go to the **Java** node, here you can tell JDeveloper to generate the Java files for each of the components, this can be helpful when you want to override any of the methods that the objects provide. For example you can indicate that you want to generate the setter and getter method and then you can add business logic to the setSalary method. Since our validation is a simple one we can use meta-data to define it instead of code.
9. Go to the **Validation** node choose the **Salary** attribute and click the **New** button to create a new validation rule
10. Choose **RangeValidator** from the **Rules** list.
11. Type a **minimum** value of **0** and a **maximum** value of **99999**.
12. Type in an **Error message** that will be displayed in case the value is not valid something like "Salary must be between 0 and 99,999".
13. Click OK in the dialog and OK again to return to the diagram and save your changes.

Rename your diagram and Click **OK**.

Generating the Application Module

Oracle ADF Business Components is based on two layers that separates the persistence layer (entity objects) from the data access layer (view objects). So far we worked with the Entity object. The Entity objects are responsible for communicating with the database directly. On top of the Entity objects Oracle ADF Business Components provide View objects. View objects are the interfaces that the developer will work with. This separation provides further reusability and maintainability. An Application Module is a collection of View objects that usually correspond to a use case of your application.

The next step for us is to create the rest of the layers to complete the Oracle ADF Business Components layer.

8. **Right Click** on an empty area of the diagram and choose **Generate->Default Data Model Components for Diagram**.
9. Go through the wizard's steps accepting all the default values.
10. In the Application Navigator expand the new **HRapp** node to see the view objects and the application module definition.
11. We want to hide some of the fields from the end user so we'll edit the View object. **Double click** on **EmployeesView** to edit the view.
12. Click the Attributes node and from the list of available attributes shuttle the JobId and CommissionPct back to the left.
13. Now lets add some formatting tips to one of the attributes. **Expand** the **Attributes** node and click **HireDate**.
14. In the **Control Hints** tab set the **Format Type** to **Simple Date**.
15. In the **Format** field write **MM-dd-yyyy**
16. Set the **Label** T**ext** to **Hired**.
17. Click **OK** to exit the dialog.

Testing the Business Service Layer

Now that you finished defining the Oracle ADF Business Components layer it would be nice to test it and see that all the definitions you made actually work. This is easy to do using the Application Module tester. A simple Swing based graphical user interface.

1. **Right Click** on **AppModule** and choose **Test**.
2. Accept all the defaults and click **Connect**.
3. A Java application appears, you can use it to work with your application module. **Double click** on the **EmpDeptFkLink1** and you'll see a simple master detail window.
4. Browse through the various departments. Note how the relationship is automatically managed for you by the ADF Business Components. You will also notice that the HireDate field is displayed according to the properties you set before.
5. Try to update the Salary field to "-8" and you'll notice the error message you specify when you try to leave the field. Correct the value to a valid one.
6. Exit the application and save all your work.

In this section we developed the Business Services layer and ran a simple client server application to test it.

Next we'll work on another type of user interface for the same Business Service – an HTML based user interface

Creating The View and Controller Layers

Creating a Databound JSP Application

In this section we'll create a simple application that manipulates the data exposed by the Oracle ADF Business Components business services layer. The application will enable users to browse departments and update an employee's details.

We'll use Struts to control the flow between two JSP pages that bind to our business services via the Model layer. Oracle ADF delivers a very easy way to bind the Control and View Layers to Business Services of any type through its Model abstraction layer. This innovative architecture is the base for JSR-227.

Departments Browser

First we want to let the user browse the departments.

To do this we'll create a struts page flow and add some data pages and define the flow between them using Strtuts.

1. Right Click on the **ViewController** node in the application navigator and choose **Open Struts Page Flow Diagram**.
2. From the component palette choose a **Data Page**, click to place it in the diagram area and call it **/browseDepts**.
3. **Double click** on **browseDepts** to edit the page, accept the default JSP page name. And click **OK**.
4. You are now in the Visual JSP editor, **add a heading** to your page "Browsing Departments" and format it.
5. Now we'll use the Data Control Palette to add database interaction to this page. Make sure the palette is viewable **Ctrl+Shift+D**.
6. Locate your cursor on the **Departmentsview1** object. At the bottom of the Data Control Palette you'll see a **Drag and Drop As** list. Choose **Read Only Form** from the list and drag the on to the page. This creates a form with an HTML table that will display data from the Departments data object.
7. Choose **Departmentsview1** object in the Data Control Palette and from the **Drag and Drop As** list choose **Navigation Buttons** and drop it inside the form that was created just after the HTML table. This will create a set of buttons that will let you scroll through your data.
8. Now we'll add the option to see the employees in each department. One of the benefits of Oracle ADF Business Components is the automatic management of master-detail relationships.

9. In the Data Control Palette expand **Departmentsview1** node so you'll be able to see the fields of this table, right after the field you'll see the **EmployeesView2**.
10. Click on **EmployeesView2** and from the **Drag and Drop As** list choose **Read-Only Table**and drop it beneath the navigation buttons.
11. Go back to the **page flow diagram** for the struts-config.xml file, **right click** on the **browseDepts** and choose **Run**.
12. In the running application browse through the departments using the navigation buttons

Notice that you didn't manually write any code to bind your JSP page to your data model. Oracle ADF takes care of binding the data to the controller automatically through the use of special data actions connected to the data pages.

Another thing to note is the formatting and heading of the Hiredate column in the employees details, the definitions you made in the model layer are reflected in the HTML user interface.

Editing Employee's Information

The next step is to allow the user to choose a specific employee and edit its data. To do this we'll create a new data page.

1. Place a **Data Page** on the page flow diagram and rename it to **/editEmp**.
2. **Double click** the **editEmp** data page to edit its HTML and add a page heading "Editing Departments".
3. From the Data Control Palette choose the **EmployeesView2** object and from the **Drop As** list choose **Input Form**. Drag and Drop the Departments object onto the HTML page.

Until now we used the drag and drop data binding from the model to the JSP only for complete records, but this operation can also be done at the item level. For example let's make sure that users can't update the department id of an employee by replacing that text field with a display field.

4. In the page editor select the **departmentId** text field and **delete** it.
5. From the **Data Control Palette expand** the **EmployeesView2** and click the **DepartmnetID** field.
6. From the **Drag and Drop As** list choose **Value** and drag and drop it to the empty space where the text field used to be.

Now we need to link back from the edit mode to the view page.

7. Back in the page flow diagram Create a **Forward** link from the **browseDepts** to **editEmp**.

8. Create a reverse Forward link from **editEmp** to **browseDepts** and **rename** it from Success **to Submit** (this can be done in the property inspector).

Next we'll enable the user to select a specific employee from the list shown for each department. We'll use a built in operation that the data controller offers us to select a specific record in a multi record set.

9. Double click on the **browseDepts** page to edit its JSP.
10. Delete the **${Row.currencyString}** tag in the page (first column of the last line).
11. **Expand EmployeeView2** in the Data Control Palette so you'll see the fields and the **Operation** nodes. Expand the **Operations** node.
12. Select the **setCurrnetRowWithKey (String)** operation and from the **Drag and Drop As** list choose **Find Row Link** and drop it **into** the **empty cell** you created. This is a special action provided by Oracle ADF Business Components.
13. **Right click** on the **Select** link that was created and choose **Edit Tag**. Change the tag **Text** to **Update**. Note the name of the event that the URL direct to – it uses the supplied **setCurrnetRowWithKey** event**.**
14. Back in the page flow diagram, **rename** the browseDepts to editEmp **forward** link to **setCurrnetRowWithKey**. This directs Struts where to go to when this event is invoked.
15. **Right click** on **browseDepts** and choose **Run**.

Browse the departments, choose one of the employees and hit the update link to try and update its data. Try to insert an invalid value for the Salary field (like -9) and watch the error message we defined in the data model show up.

Adding Transaction Management

One missing part in our application is transaction management that will let the user commit or rollback his changes. Oracle ADF makes it very easy to add such functionality using pre-built actions. Transaction management is managed at the Oracle ADF Business Components layer.

1. Open the **/browseDepts** datapage in the visual editor.
2. **Click** the **Last** button in the navigation bar **right click** and choose **Table->Split Cell**. Split the cell to 3 columns to create additional two empty cell.
3. In the **Data Control Palette collapse** the tree so only the first drill level is visible. Expand the **Operations Node**.
4. Choose the Commit Operation. Drag and drop it into the empty cell you created right next to the Last button. This will add a commit button.

5. Repeat the last step with the Rollback operation adding it to the other empty cell.
6. Click save-all and re-run your application from the page flow diagram to see the changes.

Note how at first the commit and rollback button are disabled and they only become active after you modify some data.

Creating Another Type of View Layer

Oracle ADF lets you plug and play different technologies in each of its layers. In this section will use Oracle ADF UIX to create another View layer on top of the Oracle ADF Business Components based business services layer. Oracle ADF UIX is a user interface development framework; its main benefits include an extensive set of graphical HTML components and an easy way of defining a page structure. ADF UIX will evolve to become the Oracle implementation of JavaServer Faces (JSF), and will offer a rich set if JSF components.

Building a UIX master detail form

We'll build a simple master detail form. To do this we'll create a new data page.

1. Place a **Data Page** in the diagram and call it **/masterDetail**.
2. **Double click** to edit the content of the **masterDetail** page, in the dialog that pops up make sure to **choose** the **masterDetail.uix** option from the list.
3. The page will be opened in the UIX visual editor. Go to the Data Control Palette and expand the model to see the DepartmentsView1 and its fields.
4. Notice that the model contains the master detail relationship and shows you the EmployeesView2 under the departments object. Click on **EmployeesView2** and from the **Drag and Drop as** list choose **Master Detail Many to Many**. Drag EmployeeView2 onto the middle of the page.
5. In the page flow diagram **right click** the **deptEmpDA** data action and choose **Run**.

A page that let you browse the departments and employees is displayed. Let's explore some of the unique features that this simple ADF UIX page provides.

- ADF UIX automatically provides you with range browsing for both the departments and employees. So you can scroll 10 records at a time, and even jump to a specific record range.
- Select different departments on the page to view their employee's details. Notice that when you change a department and the new department employees get displayed – only the employees part of the page is rendered from scratch and not the whole HTML page – this unique ADF UIX feature is called Partial Page Rendering.
- Click on the heading of the different columns in the tables to order the records by the specific column.

These are only some of the features you get with the rich set of user interface components that Oracle ADF UIX offers. The ADF UIX look and feel can be customized to suite your needs. Let's use another look and feel for our page.

Changing the Application's Look and Feel

1. In the application Navigator expand the node ViewController->Web Content->WEB-INF and double click to open the uix-config.xml file.
2. Near the end of the line change the line:

   <look-and-feel>**blaf**</look-and-feel> to

   <look-and-feel>**minimal**</look-and-feel>
3. Run your application again to see the different look and feel.

Deploying the Application

Deploying an Oracle ADF Application to Oracle Application Server

Installing Oracle Application Server Containers for J2EE

Oracle offers a light-weight yet powerful J2EE container known as OC4J. You can download OC4J from http://otn.oracle.com/products/oc4j. In the next section we'll be using OC4J 10g (9.0.4). First lets install the J2EE container.

1. Start a **command line** window and **run** the following script in it **[jdev-root]\jdev\bin\setvars.bat**. This scripts sets the environment variables needed to run your application correctly.
2. Unzip the OC4J.zip file to a directory for example c:\oc4j. From now on we'll refer to this directory as [oc4j-root]
3. Go to [oc4j-root]\j2ee\home and execute the command **java –jar oc4j.jar –install.**
4. You'll be prompted to insert the password for the admin user (twice).
5. Now start OC4J using the command **java –jar oc4j.jar** .

Installing Oracle ADF Runtime

In order to run ADF based applications on J2EE servers you need to install the Oracle ADF runtime libraries first. In the next section we'll step through the installation process on the Oracle Application Server Containers for J2EE (OC4J)– the process is similar on other J2EE servers as well.

1. From the **tools** menu choose **ADF Runtime Installer->Standalone OC4J**.
2. In step 1 locate your OC4J root directory (C:\oc4j).
3. Step through the wizards steps and click Finish.
4. **Stop and restart the OC4J** for the changes to take effect.

Creating an OC4J Connection

1. In the connection tab **right click** the **Application Server** node and choose **New Application Server Connection**.
2. In step 1 **name** the connection **OC4J** and select the **Standalone OC4J** connection type.
3. In step 2 fill in the **admin password** and check the **Deploy Password** option.
4. In step 3 update the **Local Directory where admin.jar for OC4J is installed** to **[oc4j-root]\j2ee\home**.
5. Click next and test your connection.
6. Click Finish.

Packaging the Application

The standard way to deploy J2EE applications with Web interfaces is through the use of WAR files. JDeveloper has wizard support for creating all the types of J2EE deployment files.

1. **Right click** on the **ViewController** node in the application navigator and choose **New->Deployment profiles->WAR file**.
2. Update the **deployment profile name** to **hr** and click **ok**.
3. In the **Deployment Profile Properties** wizard in the General node choose the **Specify J2EE Web Context Root** radio button and write **hr** as the value.
4. Accept all the other default in the profiler editor window and click **ok**.
5. In the application navigator **right click** on the **hr.deploy** file and choose **Deploy to -> OC4J**.
6. Test your application by accessing the URL http://127.0.0.1:8888/hr/browseDept.do

Deploying an Oracle ADF Application to JBoss

Installing Oracle ADF Runtime

In order to run ADF based applications on J2EE servers you need to install the Oracle ADF runtime libraries first. In the next section we'll step through the installation process on the JBoss server – the process is similar on other J2EE servers as well.

1. Start a **command line** window and **run** the following script in it **[jdev-root]\jdev\bin\setvars.bat**. This scripts sets the environment variables needed to run your application correctly.
2. From the same command line window **start JBoss** [jboss-root]\bin\run.bat
3. From the **tools** menu choose **ADF Runtime Installer->JBoss**.
4. In step 1 locate your JBoss root directory (C:\jboss-3.2.3).
5. Step through the wizards steps and click Finish.
6. **Stop and restart the JBoss** server for the changes to take effect.

Creating a JBoss Connection

1. In the connection tab **right click** the **Application Server** node and choose **New Application Server Connection**.
2. In step 1 **name** the connection **JBoss** and select the **JBoss 3.2.x** connection type.
3. In step 2 use the browse button to locate your deployment directory (C:\jboss-3.2.3\server\default\deploy).
4. Click Finish.

Packaging the Application

The standard way to deploy J2EE applications with Web interfaces is through the use of WAR files. JDeveloper has wizard support for creating all the types of J2EE deployment files.

1. **Right click** on the **ViewController** node in the application navigator and choose **New->Deployment profiles->WAR file**.
2. Update the **deployment profile name** to **hr** and click **ok**.
3. In the **Deployment Profile Properties** wizard in the General node choose the **Specify J2EE Web Context Root** radio button and write **hr** as the value.
4. Accept all the other default in the profiler editor window and click **ok**.
5. In the application navigator right click on the hr.deploy file and choose Deploy to -> JBoss.
6. Test your application by accessing the URL http://127.0.0.1:8080/hr/browseDept.do

WEB SERVICES APPLICATION

In the following section we'll guide you through integrating a Web service into your application. Oracle JDeveloper makes it easy to consume existing Web services as well as to develop and publish new Web services from any Java class or PL/SQL stored procedure.

Web Services based Business Services

In this section we'll use an existing Web service to create a simple application.

The Web service will function as a business service, and the ADF Model layer will let us use it in the same way we worked with the database based business service. The interesting thing to note is the similar development experience you get with Oracle ADF.

Note: this section requires Internet access. If you are using proxy to connect to the Internet make sure to set this in **the tools->preferences->Web Browser and Proxy settings** menu option.

Consuming an existing Web service

First we'll locate an existing Web service by browsing a UDDI repository

1. Create a **new application** and choose the **Web Application [JSP, Struts, EJB]** technology stack for it.
2. Switch to the Connections Navigator Ctrl+shift+O.
3. Expand the UDDI node and continue through **Microsoft Public UDDI->Xmethods->Services->Xmethods Delayed Stock Quotes->htpp…->Xmethods Simple Stock Quote**.
4. Right click on **Xmethods Simple Stock Quote** and choose **Create Data Control**.

The steps you just followed created a java class that acts as a data control based on the existing Web service. You can look at the code for this class to see the calls to the Web service. The next few steps will simply create a Web application that will activate the Web service pass a parameter to it and get a value back.

Building the view and controller layers

1. In the application navigator right click the **ViewController** node and choose **Open Struts Flow Diagram**
2. From the component palette add one **Data Action (dataAction1)**, One **Data Page (dataPage1)** and one **Page (untitled1.jsp)** to the diagram.
3. Create a **Forward** link from the **Data Action** to the **Data Page.**

4. Double click the Data Page to edit the JSP it and write “The stock value is:”
5. switch to the Data Conrol Palette and expand all the levels up to the methods **Return** value. **Drag** the **Return** on to the HTML page.

Now we just need to set up the first page to call out to the Web service and pass a parameter.

1. In the page flow diagram **Double click** on the **regular page** icon to open the page in the visual editor.
2. Write some text for your page heading like “Stock Application”.
3. Change the **component palette** view to see **Struts HTML** components.
4. Place the cursor under the text and from the component palette choose **Form.**
5. In the dialog that pops up set the **action** property to **dataAction1.do** and click **ok**.
6. Change the **component palette** view to see **HTML** components.
7. While the cursor is still placed inside the Struts Form, choose **Text Field** from the HTML component palette to insert it into the Form section.
8. From the component palette **insert** a **Submit Button** right after the text field.
9. Click on the **Text Field** you created and in the property palette change its **name** property to **stock.**
10. Click on the **Submit Button** you created and in the property palette change its **value** property to **Get Quote.**

The next step will make sure that the action that is activated will activate the Web service.

1. Back in the page flow diagram right click on an empty area and choose **Diagram->Refresh Daigram From all Pages**.
2. From the Data Control Palette choose the **getQuote(String), drag and drop** it onto the dataAction1 icon in your **diagram**.
3. We need to pass a parameter to this method. We can set this easily using the Structure Window.
4. Expand the Struts Config -> Actions Mapping -> /dataActions1.
5. Click on **paramNames[0]**, in the **property inspector**, change the value property to **${param.stock}**.
6. In the diagram **right click** on the **untitled1.jsp** page and choose **Run**. As a stock symbol write ORCL and press the button. You can check this out with other symbol as well.

EJB APPLICATION

The following section will guide you through the steps involved in creating a couple of Entity Enterprise Java Beans, a session façade and DTO design patterns for them and a JSP application that uses these to access data in the database.

Entity EJB is another way to create Java objects that interact with a database. There are several design patterns that you should implement when working with Entity EJB. The session façade and Data Transfer Object (DTO) are two of the more important.

As you will see in this section you use the same modeling and data binding technique you used when working with Oracle ADF Business Components when you work with EJBs as your business service layer.

EJB based Business Services

Reverse Engineering CMP EJB

In this section we'll create a couple of EJB by reverse engineering the structure of tables in the database.

1. In the Application Navigator, **right-click** the **Applications node** and choose **New Application Workspace**.
2. Change the **Application Name** to **HrApplication**.
3. In the **Application Templates** field, choose **Web Application [JSP, Struts, EJB]** from the drop down list. Click OK on the Create Application Workspace dialog.
4. In the Application Navigator, **right-click Model** (under the HRApplication node) and choose **New** .
5. In the New Gallery, under the Categories list, expand Business Tier, and select **Enterprise JavaBeans**. In the Items list, select **EJB Diagram**. Click **OK**.
6. Accept the default package name, but change the **name** of the diagram to **EJB HR Diagram.**
7. In the Navigator, click the **Connections** tab. This will bring the Connection Navigator to the front.
8. In the Connection Navigator, expand the nodes for **Database** -> **HR ->Tables**. (If you are missing the HR connection, follow the database

connection setup steps in at the start of the Oracle ADF Business Components section).

9. Click on **DEPARTMENTS** and then **Ctrl-click EMPLOYEES**. **Drag and drop** both onto the blank **EJB diagram**.
10. In the Create from Tables dialog, select **EJB 2.0 Entity Beans** and click **OK**.

You now have two entity EJBs in your diagram. The code that implements these EJBs was automatically generated for you. You can access it by clicking on the EJB objects in the application navigator and then clicking on the Java files that appear in the structure window beneath it.

Creating a Session Facade

Session Façade is the most widely used of all EJB design patterns. Session Facade allows you to properly partition the business logic in your system to help minimize dependencies between client and server, while forcing use cases to execute in one network call and in one transaction. In this step you will add a local reference from a session bean to the entity beans.

11. In the **Component Palette** (View | Component Palette menu option), click **Session Bean** and then **click** inside the EJB **diagram**.
12. The EJB wizard welcome page appears click **Next**.
13. On the Name and Options page, change the **EJB Name** to **hrApp** (case senstive). Click **Next**.
14. On the Class Definitions page, accept the default values and click **Next**.
15. On the EJB Home and Component Interfaces page **select** the box for Include **Local Interfaces**.
16. Click Finish.
17. In the **Component Palette** choose **EJB Local Reference**.
18. In the **diagram**, click on the **session bean** (hrApp), **drag** to the **entity bean** (Departments) and click again.
19. The local reference is displayed as a line between hrApp and Departments.

Implementing DTO Design Pattern

Data Transfer Object design pattern provides better maintainability by separating use cases and data object model (entity beans), reuse entity beans across different applications, and increases performance when the attributes from multiple entity beans can be passed to the client within one call.

20. Select the **Departments** bean in the EJB diagram.

21. **Right click** select **Generate | Data Transfer Object**.
22. Select the **Employees** bean in the EJB diagram.
23. **Right click** select **Generate | Data Transfer Object**.
24. You can rearrange the diagram if needed.

Adding Business Logic

1. On the EJB diagram, **click** in the **first empty cell** of the hrApp **session bean** and add a new attribute **showDepartments : Collection** (note case sensitivity).
2. On the diagram, **right click hrApp** and choose **Go to Source | hrAppBean.java Bean Class**.
3. The Code Editor opens.
4. In the Code Editor, scroll down to the getShowDepartments() method.
5. In the Code Editor, modify this method so that it returns a collection of Department DTOs:

```
public Collection getShowDepartments()
{
try
{
ArrayList al = new ArrayList();
Collection col = this.getDepartmentsLocalHome().findAll();
DepartmentsLocal dept;
Iterator it = col.iterator();
while(it.hasNext())
{
DepartmentsLocalDTO deptsDTO = new
DepartmentsLocalDTO((DepartmentsLocal)it.next());
al.add(deptsDTO);
}
return al;
}
catch (FinderException e)
{
System.out.println(e.toString());
throw new javax.ejb.EJBException(e);
}
catch (NamingException e)
{
System.out.println(e.toString());
throw new javax.ejb.EJBException(e);
}
}
```

6. Import the following (You can use the Java importer help tips with Alt+Enter)

```
import java.util.ArrayList;
import java.util.Iterator;
import javax.ejb.FinderException;
```

7. Compile the code by right-clicking hrApp in the Navigator and choosing Make hrApp.

Creating a Model from the EJB

Now that we have the EJB set up and implementing some design patterns we want to create a data model from it that we can use with Oracle ADF to develop a user interface.

1. Select the **hrApp** bean in the application navigator **right click** and choose **Create Data Control**.

Currently the data control knows that it's supposed to work with a collection, but doesn't know what kind of objects that collection will consist of. We specify the bean class by setting a new property in the hrAppLocalDataControl.xml file. We'll see how JDeveloper lets us edit an XML file using visual tools.

2. Select the **hrAppLocalDataControl.xml** file in the application navigator.
3. In the **structure window** click the **showDepartment** attribute.
4. In the property inspector change the Bean Class property to **model.DepartmentsLocalDTO**.

Creating JSP interface to the EJB

Now the EJB functions like any other business service for our Model layer, it is very simple to build a View layer for it now.

1. Right Click on the **ViewController** node in the application navigator and choose **Open Struts Page Flow Diagram**.
2. From the **component palette** choose **Data Page** and click in an empty space in the diagram. Rename the page to **/dept**
3. **Double click** on the **/dept** page to edit its JSP content in the visual editor
4. From the **Data Control Palette** select **showDepartments** and from the **Drop As list** select **Read Only Table** drag and drop it on the empty page.
5. Back in the page flow diagram **right click** on **/dept** and choose **Run**.

COMPLETE LIFE CYCLE AND OPEN SOURCE TOOLS

Open Source tools and frameworks

One of the great things about the Java universe is the many frameworks and tools available for Java developers. Many of these frameworks and tools are open source and free. In addition to using Struts, Oracle JDeveloper integrates three of the leading open source developer's tools:

- JUnit – a Java based code testing framework
- CVS – a software configuration management tool
- Ant – a build tool

In the following section you'll see how the integration of these frameworks and tools inside Oracle JDeveloper provides you with full life cycle support.

Using CVS

In this section we'll create a simple project and see how we can use the open-source Current Version System (CVS) tool to manage the life cycle of the files in this project. First let's set up CVS on our machine and in JDeveloper.

Setting-up CVS

1. Download and install CVS 1.11 from http://www.cvshome.org/.
2. Make sure that the CVS directory is included in your system PATH variable. Check this by issuing the following command from the command line:

   cvs -version

3. Create a new directory on your file system we'll use c:\jdver in this sample.
4. Initialize a new repository from the command line using: cvs –d c:\jdver init
5. Startup JDeveloper.
6. Set JDeveloper to use CVS as its version control tool. From the **versioning menu** choose **Select System** , check the **CVS radio button** and click **ok**.
7. Switch to the connection tab **Ctrl+Shift+O**, **right click** on the **CVS Server** node and choose **New CVS Connection**.
8. In the wizard go to step 1 and **name** the connection **money**.
9. In step 2 keep the **access method** to **Local**, for the **Repository path** insert your directory **c:\jdver**.
10. Click next to go to step 4 and **test** your connection. Make sure you received **Connection test successful**. Click **Finish**.

Creating the Money application

In this section we'll create a simple money class that we'll work with in the rest of the open source section.

1. In the Application Navigator, **right-click** the **Applications node** and choose **New Application Workspace**.
2. Change the **Application Name** to **Money** the **Application Package Prefix** should be **money** and the **Application Template** should be No Template [**All Technologies]**.
3. **Right click** on the **Project** node in the application navigator and choose **New** from the **General** category choose **Java Class**.
4. Name the class **money** and click **ok**.
5. In the editor that gets open update the class code to be:

```
package money.mypackage;
public class money
{
 private int fAmount;
 private String fCurrency;

   public money(int amount, String currency) {
      fAmount= amount;
      fCurrency= currency;
   }

 public int amount()
 {
  return fAmount;
 }

 public String currency()
 {
  return fCurrency;
 }

   public money add(money m) {
     return new money(amount()+m.amount(), currency());
}

 public boolean equals(Object anObject) {
   if (anObject instanceof money) {
      money aMoney= (money)anObject;
      return aMoney.currency().equals(currency())
         && amount() == aMoney.amount();
   }
   return false;
}
}
```

6. **Right click** the editor and choose **Make** to compile the code.

We've created a new project with a class. Now lets upload the project into CVS.

Managing changes with CVS

In this section you'll see the basic work with CVS inside JDeveloper and how you can easily manage versions and compare them.

1. In the Application Navigator, **right-click** the **Project** and choose **Import Module**.
2. In step 1 set **connection** name **money**. All the other values in the wizards should be kept to their default so click **Finish**.
3. The Client node in your application navigator now has a little pearl icon indicating that it is now checked out.
4. Expand the application navigator and double click **money.java** to **edit** its **source**.
5. Add a remark to the source:
   - **/*this class implements the money object*/**
6. **Save** the file and note the change in the icon for the file showing a star indicating the file has changed.
7. **Right click** the **money.java** file in the application navigator and choose **Compare With -> Previous Revision**.
8. In the compare window you can see the changes to the file clearly indicated.
9. **Right click** the **money.java** file in the application navigator and choose **Versioning -> View History**. This shows you the history of the file.
10. **Right click** the **money.java** file in the application navigator and choose **Versioning -> Commit.** Insert a comment saying "remark added" and click ok. Your file was checked in to the repository.

Building Test Scripts with JUnit

JUnit is an open-source testing framework for Java. You use JUnit to test your code to make sure it is functioning properly. In the following section we'll use JUnit to test the money class we have just created. This section is not aimed at teaching JUnit but rather show you the way you use it from inside JDeveloper.

Creating a test fixture

First we'll create a text fixture containing variables that will be used in two tests.

1. In the Application Navigator, **right-click** the **Project** node and choose **New** and choose **General->Unit Tests(JUnit) -> Custom Test Fixture**.
2. Accept all the defaults in the wizard and click **Finish**.
3. Add the following import

```
import money.mypackage.money;
```

4. Add the following two lines at the top of the TestFixture1 class code

```
public money f12CHF;
public money f14CHF;
```

5. Change the setup method to be:

```
public void setUp()
{    f12CHF= new money(12, "CHF");
     f14CHF= new money(14, "CHF");
}
```

6. Right click in the editor and choose Make to compile your code.

Creating simple tests cases

In the following steps we'll define two tests to test the add and equals method of the money class.

7. In the Application Navigator, **right-click** the **Project** node and choose **New** and choose **Unit Tests(JUnit) -> Test Case**.
8. In the wizard step 1 click **browse** to choose the **money.client.money Class**.
9. **Expand** the **money.client.money** methods and **check** the **add** method to generate test cases for them. Click **Next**.
10. Accept all the defaults in Step 2. Click **Next**.
11. In **step 3** of the wizard click the **Add** button and add the **money.mypackage.test.TestFixture1** test fixture. Click **Finish**.
12. A new test case has been created for you.
13. Add the test code. Modify the **testadd** method:

```
public void testadd()
{
  money expected= new money(26, "CHF");
  money result= fixture1.f12CHF.add(fixture1.f14CHF);
  assertTrue(expected.equals(result));
}
```

14. Right click and choose make to compile your test case.

Now we'll add another test case for the equals method.

15. In the Application Navigator, **right-click** the **Project** node and choose **New** and choose **Unit Tests(JUnit) -> Test Case**.

16. In the wizard step 1 click **browse** to choose the **money.client.money Class**.
17. **Expand** the **money.client.money** methods and **check** the **equals** method to generate test cases for them. Click **Next**.
18. Accept all the defaults in Step 2. Click **Next**.
19. In **step 3** of the wizard click the **Add** button and add the **money.mypackage.test.TestFixture1** test fixture. Click **Finish**.
20. A new test case has been created for you.
21. Add the test code. Modify the **testequals** method:

```
public void testequals()
{
  assertTrue(!fixture1.f12CHF.equals(null));
  assertEquals(fixture1.f12CHF, fixture1.f12CHF);
  assertEquals(fixture1.f12CHF, new money(12, "CHF"));
  ssertTrue(!fixture1.f12CHF.equals(fixture1.f14CHF));
}
```

22. Right click and choose make to compile your test case.

Creating and running a Test Suite

Once you have one or more test cases you need to create a test suite that you can run with JUnit.

1. In the Application Navigator, **right-click** the P**roject** node and choose **New** and choose **Unit Tests(JUnit) -> Test Suite**.
2. In the wizard accept the defaults for Step 1, In **step 2** click the **Add** button browse the money package and add the both **moneyTester1** and **moneyTester2** test cases.
3. Complete the wizard and click **Finish**.
4. In the Application Navigator, **right-click** the **AllTests1.java** node and choose **Run**. JUnit will run your test and everything should work fine. You can see the results of the two tests in the Test Hierarchy tab.

Changing the code and retesting

Now lets change something in the code and see if our test will catch the problem.

5. In the Application Navigator, **double-click** the **money.java** file to edit it.
6. Change the add method to return the following:

   return new money(1+amount()+m.amount(), currency());

7. Compile your code.

8. In the Application Navigator, **right-click** the **AllTests1.java** node and choose **Run**. JUnit will run your test and this time the testadd test will fail (indicated by the red bar). You can see the results of the two tests in the Test Hierarchy tab.

9. **Right click** the **money.java** file in the application navigator and choose **Compare With -> Previous Revision**. You can use the right mouse context menu to navigate to the line that was changed, so you can view where the problem started. And you can fix the code back to its previous version.

Using Ant

Ant is a build tool that is used to create batch processes to manage your project. In this section we won't teach you Ant, but we'll show you how to use Ant from inside JDeveloper.

Creating Ant tasks

First lets create an Ant task and run it.

1. In the Application Navigator**, right-click** the **Project** node and choose **New ->General->Ant->Build File from Active Project Profile.**

2. Accept all the defaults in the dialog window and click **ok**.

3. A new build.xml Ant file was created. You can browse its structure using the structure window. You'll see that there are several tasks (targets) that were created by default these include tasks to compile the code and create javdoc. You can see the properties of each target in the property palette.

4. **Right click** on the **build.xml** file and choose **Build Default Target**.

Adding and Ant Target

Now we'll add a new target to our Ant file that will test our project after its compilation.

5. Edit the build.xml file and add the following text:

```
<target name="unit.test" depends="compile">
  <java classname="junit.textui.TestRunner"
      classpathref="classpath"
      dir="."
      fork="true">
    <arg value="-noloading"/>
    <arg value="money.mypackage.test.AllTests1"/>
  </java>
</target>
```

6. Now you can run this specific target from JDeveloper. **Right click** on **build.xml** and choose **Build Target->unit.test**.

7. The test should execute and the results will be shown. Note that we are using a tester class that has a text output and not a graphical output.

MORE INFORMATION

Additional information on OracleJDeveloper is available on Oracle Technology Network (OTN) at http://otn.oracle.com/products/jdev/ .

The website provides online demos, white papers, technical notes, samples, tips and trick, and a discussion forum for Oracle JDeveloper. Make sure to visit this site to maximize your development experience and to get the latest JDeveloper news.

Oracle JDeveloper 10g Reviewers Guide
March 2004

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
www.oracle.com

Oracle Corporation provides the software that powers the internet.