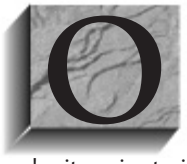
# CHAPTER 1

# Overview of Oracle JDeveloper 10*g*

*Software is like entropy. It is difficult to grasp, weighs nothing,*
*and obeys the second law of thermodynamics;*
*i.e., it always increases.*

—Norman R. Augustine, Chairman, Exec. Comm.,
Lockheed Martin Corporation



racle JDeveloper 10*g* (JDeveloper) is an integrated development environment (IDE) for Java programming. It offers a rich set of features for designing, developing, debugging, and deploying Java and other related files that are part of the Java 2 Platform, Enterprise Edition (J2EE) strategy. JDeveloper is a development framework containing many wizards and code generators that make it easier to implement complex functionality in Java, enabling you to concentrate on solving business problems. It also offers strong code organization and configuration management.

The J2EE environment is quite complex, and JDeveloper strives to support the management of all of the different components including JSP pages, UIX applications, Struts, and web services. The IDE assists with XML editing and data source access and manipulation.

This part of the book provides an overview of the features and functions of JDeveloper 10*g*. This chapter offers an introduction to the main concepts needed to work with JDeveloper and includes some simple hands-on practices to give you a quick head start building applications in JDeveloper 10*g*. Chapters 2 and 3 describe the main work areas and tools of the Integrated Development Environment (IDE). Chapter 4 introduces the Oracle Application Development Framework (ADF), a rich technology environment that supports all facets of J2EE development. For those new to Java, Chapter 5 provides an overview of the Java language concepts needed to work effectively with JDeveloper. Chapter 6 discusses the importance of consistent naming conventions and provides suggestions about how to name the elements needed to develop applications with JDeveloper. Finally, Chapter 7 includes further details about J2EE architectures and outlines approaches to deploying J2EE applications.

# JDeveloper: Past, Present, and Future

JDeveloper's roots go back to 1997, when Oracle licensed the JBuilder Java-development tool from Borland International to integrate it with Oracle's databases and applications tools for both Internet and traditional client/server platforms. At the time, Borland's JBuilder was a strong Java development tool. Purchasing the rights to the JBuilder source code allowed Oracle to jump-start its entry into the Java development environment. The initial JDeveloper 1.0 release (called "AppBuilder for Java") in 1998 was quite close to its JBuilder foundation. Later in 1998, it was renamed JDeveloper. The similarity between JBuilder and JDeveloper continued through the 2.0 releases in 1999. In these early versions, you can see the tool's maturation in the Java environment. While releasing only cosmetic changes to the product, behind the scenes Oracle was working on elegantly solving the problem of Java programs connecting to relational database objects.

## Past: Product History and Roots

While JDeveloper releases 1.0 and 2.0 were useful Java development products, they provided little support for building applications that would interact with an Oracle database. To be fair,

this was the state of the art at that time. Hardy C++ and Java programmers routinely took up the task of writing lots of code to access Oracle databases. Unfortunately, this coding required a great deal of effort, even on the part of a skilled programmer. Therefore, the early JDeveloper users were primarily Java developers who were looking for ways to create applications that would interact with Oracle databases. Oracle professionals who were accustomed to products that interacted easily with the database and who built applications efficiently using tools such as Oracle Forms Developer did not rapidly adopt this new product.

Release 3.0 of JDeveloper introduced Business Components for Java (BC4J), the forerunner of the Application Development Framework Business Components (ADF BC) included in JDeveloper 10*g*. This gave developers an easy way to connect their applications to the database without having to write hundreds of lines of code. BC4J quickly differentiated JDeveloper from other products.

The next major release (9*i*) brought a complete rewrite of the product in Java and a new way of binding components to BC4J using JClient for local Java client code and the BC4J Data Tags Library for server-run code such as servlets and JSP files. In release 3.x, Oracle supplied its own custom components to bind to BC4J; in 9*i* bindings were made to standard components.

Although JDeveloper is now Oracle's primary development tool, Oracle offers a number of other products that enable developers to build applications and deploy them over the Web:

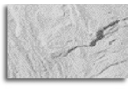
- **Oracle Forms Developer** (sometimes called "Web Forms") has continued to mature. Since the 9*i* release, the Web is the only way to deploy Oracle forms. This tool uses applet technology that has largely been abandoned by the broader development community for Internet applications due to issues related to firewalls and performance. Oracle Forms Developer applications on the Web are still strong contenders for intra-company, rich client applications that require highly responsive and interactive user interfaces.
- **PL/SQL Web Toolkit** (mod_plsql) had its origins in the early releases of the Oracle Application Server, and Oracle Designer uses it to create applications that generate HTML client code. The PL/SQL Web Toolkit allows you to write PL/SQL in the database that can output HTML to a browser.
- **PL/SQL Server Pages (PSPs)** allow you to embed PL/SQL within HTML. PSPs leverage the concept of server pages in a similar way to JavaServer Pages files.
- **Oracle Portal** was originally designed as a simple utility to allow ad hoc access to a database and was marketed as "WebDB." Portal has evolved into a useful website development and content management tool. It is not a mainstream application development tool although it has some utilities that allow you to create simple web applications.
- **HTML DB** is Oracle's latest offering for web development. It provides an easy way to create applications of low to moderate complexity that are based directly on the database and do not require an application server. Its learning curve is less steep than that of JDeveloper, but HTML DB does not provide the robustness, flexibility, or depth of functionality of JDeveloper. HTML DB is delivered as part of the Oracle 10*g* database and is designed to be even easier to use than Portal, providing extremely rapid development for simpler web applications.

### Why JDeveloper?

With all of these alternatives, why has Oracle seemingly decided to pursue JDeveloper as the primary development tool? The answer demonstrates Oracle's long-range planning strategy. Oracle JDeveloper 10*g* is built for developing J2EE applications. As such, JDeveloper sits on a strong foundation. Oracle's earlier products had to make compromises based upon existing technologies, accommodations to backward compatibility, and internal Oracle politics. JDeveloper is a development environment for J2EE, and J2EE has vast support from other vendors and the backing of recognized standards.

As JDeveloper has moved from the 1.0 release to 10*g*, it has evolved from a straightforward Java development product into a sophisticated J2EE development environment. In all prior releases, the focus seemed to be primarily on providing technical capabilities. Version 10*g* represents a maturation of both the underlying technical foundation as well as the product's ease of use. Nowhere is this more evident than in the area of web application development. The evolving WYSIWYG screen designers for JSP and ADF UIX code as well as the graphical Struts development environment provide a much more visual-graphic way to build web applications than competing products.

Oracle's direction for building Java applications to access an Oracle database became clearer through its introduction of BC4J in the 3.0 release. BC4J helped to automate most of the difficult work required to make Java code interact with relational database tables. BC4J was introduced as a framework to support Java interaction with a database. The BC4J wizards automatically generated the code necessary to allow Java applications to safely interact with the database, solving the security, locking, and performance problems that had hindered earlier efforts. The generated code utilizes an Oracle-supplied Java library. Therefore, the actual amount of code generated by the wizards is small.



**NOTE**
*Chapter 4 discusses the topic of frameworks and ADF in more detail.*

The JDeveloper 9*i* release included improved Business Components for Java. Oracle added support for Sun's Model-View-Controller (MVC) architecture (implemented in JDeveloper as the client data model), replacing version 3's Data-Aware Components (DACs) (built on Sun's InfoBus architecture, which was abandoned by the industry). The MVC architecture allowed developers to build Java applications or JavaServer Pages to access an Oracle database with much more efficiency. In the 9*i* release, Oracle's grand vision for JDeveloper was visible, indicating a much broader scope than the earlier versions. The inclusion of some Unified Modeling Language (UML) diagrams, software configuration management (SCM) integration, and the ability to generate Data Definition Language (DDL) point to Oracle's long-term commitment to this product and technologies.

## Present: Where Is JDeveloper Now?

There are two parts to the vision driving the JDeveloper product. The first is influenced by the J2EE and broader Java development communities, which strive to make JDeveloper 10*g* a full-featured J2EE development product. In this area, the JDeveloper team can take advantage of Oracle's experience with products such as Oracle Forms Developer to provide graphical or metadata-based application development, which improves the efficiency of Java and web developers. Visual editors, diagrammers, property inspectors, and component palettes have been incorporated

into JDeveloper to help generate simple applications as well as to support and manage larger and more complex ones. Some portions of JDeveloper are still evolving, specifically the WYSIWYG graphical Struts designers and visual editors, which are solid efforts that continue to mature quickly. The 10*g* release deserves high marks for supporting the J2EE/Java development vision. Building J2EE applications remains a challenging task with a steep learning curve. JDeveloper 10*g* has made it easier for development organizations to make the transition from fourth-generation language (4GL) tools such as Oracle Forms Developer or Delphi into the J2EE space. Developers currently using Oracle Forms Developer and PL/SQL cannot simply replace those tools with JDeveloper and Java to create complex client/server–style applications deployed over the Web. Building fully featured web applications that allow customers to safely and efficiently interact with the database is a much more complex task.

The second part of the vision that Oracle is pursuing is that of a unified design and development environment. Here, JDeveloper is moving toward being a single point of entry for systems analysis, design, development, and deployment. JDeveloper 10*g* includes a UML Use Case Diagrammer, which allows developers to enter use cases and generate HTML documents. The product also includes the first generation of a database modeler using UML class diagrams. The modeling portions of JDeveloper are still maturing and are not yet at the point where they can be utilized for full lifecycle systems development in the same way as Oracle Designer.

### Support for JSP Files

A JavaServer Page (JSP) file is compiled into a *servlet,* which is a pure Java program. The servlet produces an HTML stream that is sent to the browser. Though most of the code can be written in Java, the user interface (UI) portion is created in HTML. A JSP performs at least two functions: it renders web browser content (the results of Java processing) to the user, and reduces (or eliminates) logic code in the UI for the developer. As applications grow, the benefits of concise code are evident.

As a web application development tool for the building of JSP files, JDeveloper is almost a complete solution for traditional Oracle developers who are used to the simple "one-product development environment" of Oracle Forms Developer. A new visual editor allows users to build JSP pages using a representation of how the page will look. Elements can be dragged from the Component Palette and dropped into the work area. This visual design environment is somewhat similar to what 4GL developers are accustomed to; however, JDeveloper's support for the visual design of the HTML page is still limited compared with a tool such as Microsoft FrontPage.

### Support for ADF UIX Applications

Oracle created its own alternative to JavaServer Pages technology called ADF User Interface XML (ADF UIX or just UIX). UIX pages deliver a level of functionality similar to that of JSP code but are based on Extensible Markup Language (XML) tags. The ADF UIX tag library has a better visual representation and somewhat better performance than standard JSP tag libraries. For example, by default, UIX tags can be coded to only require a portion of the screen to be refreshed when only a portion of the screen is changed, resulting in a more user-friendly interface than JSP files, where the whole screen is typically refreshed. With JSP pages, you can also use HTML and JavaScript code to specify that the screen is only partially refreshed.

ADF UIX development is one of the areas where the JDeveloper 10*g* release has made great improvements. Developers can build JSP pages or ADF UIX pages with virtually the same user-friendly interface.

**NOTE**
*ADF UIX is described further in Chapter 19.*

### Support for Java Applications

JDeveloper 10*g* provides complete support for rich Java applications (Java programs running on the client machine), including visual editing and property setting in a developer-friendly interface. You can build applications of the same or greater complexity and sophistication as was possible using products such as Oracle Forms Developer. However, you will not be able to build this type of application as quickly using JDeveloper as you can with Oracle Forms Developer. The JDeveloper wizards are not sufficiently mature for JDeveloper to compete effectively with Oracle Forms Developer as a rapid application development (RAD) tool.

While Oracle Forms Developer manipulates properties of objects that are stored in an internal repository, JDeveloper is actually a code manipulator and organizer. Although you interact with the JDeveloper wizards and IDE areas as in a 4GL environment, you are really manipulating Java code. The application development process always goes beyond the capabilities of the JDeveloper wizards, and manual intervention is required to modify property settings that the wizard assigns or to add code that the wizards do not generate.

## Future: The Vision

What will the JDeveloper tool encompass going forward? Tactically, JDeveloper 10*g* is a mature J2EE development environment. Developers will be pleased with the progress since the 9*i* release. In the short run, the J2EE development environment will continue to mature and add new features that enter the technological infrastructure from standards organizations such as the Java Community Process (JCP, at www.jcp.org) and Web Services Interoperability Organization (WS-I, at www.ws-i.org). For example, JavaServer Faces technology is the latest addition to the Java platform. As described in Chapter 4, you can add JSF technology support to JDeveloper now and this support will be expanded in the near future. It is also expected that the UML modeling area will evolve rapidly toward database modeling and design so that JDeveloper will match and ultimately surpass Designer. The activity, class, and use case modelers will also evolve to allow JDeveloper to more easily support complex process-based application development.

The introduction of the Application Development Framework represents a significant rethinking of the development environment. What are still lacking are advice for development best practices and a clear System Development Life Cycle (SDLC) for building J2EE systems. To be fair, the entire industry is still waiting for a coherent SDLC to emerge for building J2EE systems based on solid relational database back-ends.

JDeveloper 10*g*'s motto, "Productivity with Choice," although reflective of the tremendous flexibility of the tool, sidesteps the additional complexity that has made traditional Oracle developers reluctant to embrace J2EE. The popularity of less flexible, but potentially more productive, alternatives such as Oracle Portal and HTML DB demonstrates that, in addition to flexibility, developers also want guidance for systems development best practices.

The expected long-range vision is that JDeveloper will move closer and closer to being a unified design and development environment. However, progress toward this goal has been slower than hoped for in the last three years, mainly due to Oracle placing emphasis on creating an excellent J2EE product. In this respect, they have succeeded admirably. The question remains as to how much attention is given to the ultimate vision vs. deployment of resources toward

tactical, technical J2EE topics. The 10*g* release represents significant improvement over 9*i* because of the following:

- Introduction of the Application Development Framework
- Expanded 4GL development support for Struts, JSP code, and ADF UIX pages, including drag-and-drop data binding to connect Java client and web client view layers with business services layers such as ADF BC and Enterprise JavaBeans (EJB)

As with all popular Java development tools, JDeveloper 10*g* is clearly evolving, making it an ever more user-friendly tool that offers substantive support for the J2EE architecture and is responding very well to changes in that architecture as they occur.

# What Is New in JDeveloper 10*g*?

The 10*g* release of JDeveloper includes some important new features and some significant improvements of other features. Some of the new additions to the product are discussed briefly in this section with references to more in-depth information contained in other portions of the book.

## New Integrated Development Environment (IDE)

There have been major changes to the JDeveloper IDE starting with the way in which applications are organized.

JDeveloper 10*g* has also provided better support for team-based development by adding to the features that integrate JDeveloper with Oracle SCM and providing additional support for external products such as Concurrent Versions System (CVS).

Chapters 2 and 3 discuss the IDE in more detail. Experienced users may be tempted to skip these chapters, but they will miss much important information and many useful tips if they do. Due to the changes in the 10*g* release, the authors recommend that even those who have used previous versions of JDeveloper read these chapters carefully.

### Technology Templates

Now when you create application workspaces, you select the underlying technology scope using a technology template. The *technology template* contains default projects that are oriented toward a certain technology such as Struts. The template affects what options and objects are available by default for use in the projects. Almost all of the lists in 10*g* are context-sensitive, so inappropriate options are rarely available. As you create applications in JDeveloper 10*g*, you need to select from one of the following default templates:

- **Web Application [Default]** This template creates a Model project for ADF Business Components and a ViewController project for Struts, ADF UIX, or JSP components and is used for building web applications using Struts that need to access ADF Business Components.
- **Web Application [Default - no controller]** This template creates a Model project oriented toward ADF Business Components and a View project for ADF UIX or JSP components and is used for building web applications that need to access business services.

- **Web Application [JSP, Struts, EJB]** This template creates a Model project for the EJB data model and a ViewController project for JSP and Struts components and is used for building web applications that need to access business services based on Enterprise JavaBeans.
- **Web Application [JSP, EJB]** This template creates a Model project for the EJB data model and a View project for JSP components (with no controller) and is used for building web applications that need to access business services based on EJBs.
- **Java Application [Default]** This template creates a Model project for ADF Business Components and a View project for the client and is used for building rich client applications that need to access ADF Business Components.
- **Java Application [Java, Swing]** This template creates a Client project for the Java, Swing/JFC, or JavaBeans source code and is used for building Java applications that are not connected to a data source.

Depending upon which template you select, different projects with different default components and properties will be available by default. You can also select the "No Template" option, which creates a single project with access to all alternatives.

The inclusion of application templates is very helpful. The J2EE environment is so rich and varied that the JDeveloper templates provide important guidance. Once you have made a decision about which technology scope to use, inapplicable options will not be available unless you explicitly allow them. This is useful in many cases where you need to go outside of the bounds of the template by turning off the filters. The New Gallery includes a *Filter By* pulldown, which is either set to "Project Technologies" (filtered) or "All Technologies" (unfiltered) so that you can access items outside the technology scope of the project.

Once you evolve your unique development style, you can create and save your own technology templates in JDeveloper, which can be applied to new application workspaces.

## Application Development Framework

The Application Development Framework is one area of JDeveloper that is very important but will not be obvious to most users. The benefits of ADF are evident when creating complex production applications. Even then, in order to truly appreciate ADF, you will need to have built other J2EE-compliant applications without it.

When building applications, there are numerous places where the framework will automatically take advantage of components in ADF. This greatly decreases the amount of coding required. Other benefits of ADF are evident when modifications to the applications are required. Traditionally built J2EE applications are notoriously difficult to modify. Things that should be simple such as adding an attribute to a table that then must be maintained in the applications can require hours, if not days, of effort without ADF. It is still not trivial to make these modifications using ADF, but it is much easier than the alternative.

ADF is discussed in detail in Chapter 4 and ADF Business Components are discussed in Part II.

## Struts

Struts is a framework that supports the logic behind building web applications. It was developed by The Apache Software Foundation (jakarta.apache.org/struts/). Struts technology was created to

bring order to the chaos of applying complex logic to the process of building and managing entire web applications including Java program files and the UI (JSP, UIX, and HTML) files.

Struts is now a popular framework for managing web application development. Despite its popularity, it has a very steep learning curve. Struts is one area where JDeveloper 10*g* has done an extraordinary job of making development much easier. The JDeveloper support of Struts through ADF greatly reduces the level of effort required to build a Struts-based application. Nowhere is the power of ADF more evident than in building Struts-based web applications.

Working with Struts still requires a great deal of skill. A thorough understanding of the Struts framework as well as Oracle's implementation of Struts is required to successfully build Struts-based applications in JDeveloper. However, once you understand the framework, using JDeveloper to build web applications is almost as rapid as the productivity that can be achieved in a 4GL. As an added benefit, the code generated is consistent and largely self-documented by JDeveloper's graphical diagramming tool, which creates page flow diagrams. Chapter 17 provides more detailed information about Struts and some hands-on practices to explore this technology.

## Modeling in JDeveloper

In addition to the great strides made in J2EE development with the 10*g* release, some progress was also made in improving the modeling capabilities of the tool. JDeveloper still does not support the entire SDLC, but it continues to move in that direction.

### Database Modeler

Developers want to be able to create models within JDeveloper. JDeveloper 10*g* introduces a physical database modeler that allows users to specify tables, columns, and foreign key relationships using UML class diagrams.

Full UML-based modeling including inheritance, aggregation, and composition capabilities would be a welcome addition to JDeveloper at some point. The 10*g* release includes the beginnings of a solid data modeling tool using a limited subset of the UML. Conspicuously absent is the notion of generalization.

The Database Modeler cannot be used yet for complete logical and physical database design. Oracle Designer should still be used for that purpose. JDeveloper users without access to a full-featured database design tool like Oracle Designer may find JDeveloper's modeling capabilities adequate for simpler applications. However, JDeveloper should not be considered a viable modeling environment for enterprise-level database construction.

### Use Case Modeler

Use cases provide a structured method for describing analysis and design elements in a system. The core elements are Actors (people, organizations, systems) and Use Cases (functions that actors perform). Use cases are typically expressed using Scenarios that represent the steps undertaken to execute the use case. Use cases can be expressed graphically, using text, or a combination as in JDeveloper 10*g*.

The Use Case Modeler included in JDeveloper 10*g* is a graphical front-end that allows users to store an entire use case diagram as a single HTML document. Text descriptions can be entered into a document resembling a Microsoft Word document that is suitable for high-level system documentation. The Use Case Modeler provides a convenient graphical user interface for including this information in your projects. However, it has not evolved sufficiently yet to support use

case–based analysis of a large project because it is too difficult to handle a large number of use cases as HTML documents without some type of repository to manage all of the information.

## Business Services

In the 9*i* version of JDeveloper, even though it was possible to have persistent storage of objects in places other than BC4J (now called ADF BC), there was a clear slant toward Oracle's business components framework. In 10*g*, there is more explicit support for alternatives to ADF BC such as EJBs, web services, and Java classes (accessed with TopLink). In most cases, ADF BC is the preferred alternative and is discussed most extensively in this book; however, some discussion of EJBs, Java classes with TopLink, and web services is included in Chapter 14.

### Web Services

Web Services are an important emerging area of web application development. They allow applications built in different development environments to be integrated. Developers can create reusable components that can either be called internally or published and made available over the Internet to anyone with the appropriate permissions.

This is an exciting new technology that is being adopted in a host of different contexts. JDeveloper has included a number of features to make the creation of web services much easier. For example, taking a PL/SQL program unit or Java class and turning it into a web service can be accomplished as a "one mouse-click" operation.

## Desupported and Deprecated Features

As with any new release, some features of old releases are not supported or are *deprecated* (currently supported but soon to be phased out). Further discussion of these features is available on OTN (otn.oracle.com), but the following sections summarize this information.

### Desupported Features

The following features do not appear in the JDeveloper 10*g* product although previous releases still support them:

- Data-Aware Controls (DAC)
- Deploying business components to a VisiBroker Object Request Broker (ORB)
- Deploying business components to an Oracle database
- Generating Java CORBA files from an Interface Definition Language (IDL) definition

### Deprecated Features

The following features are still available but are planned to be desupported in future releases. Therefore, they should not be used for new applications code:

- Data Web Beans and HTML Web Beans for JSP applications
- Business Components Data Tags (BC4J Data Tags Library) for JSP applications
- UIX JSP tags, BC4J UIX JSP tags, and BC4J UIX XML tags

In addition, the plans for future releases include desupport of SQLJ (SQL embedded in Java).

# Creating Application Code in JDeveloper

JDeveloper is an application development tool that can support your first steps in the Java world. It can act as a blank sheet of paper for the sophisticated do-it-yourselfer, or as a code generator for those who prefer developing applications by using 4GL techniques such as drag and drop. JDeveloper can also automatically generate basic database interface code, allowing you to customize the results to your heart's content.

Coming to an understanding of Java and JDeveloper is like trying to learn English and a computer word processing program at the same time. You should have some experience with other computer languages and application conventions before you leap into this type of effort. It is a good idea to actually build the items and structures in the hands-on practices presented in this book as you read the chapters. In this way, you can quickly get a feel for the development environment by creating real code.

The first major difference between the traditional environments and the JDeveloper environment is that in some development environments, the user interface and its interaction with the database are inseparable. However, in JDeveloper, each program will usually consist of two JDeveloper projects:

- **Model project** containing business services components built and written using Java and XML to provide the database interaction components (business services).
- **View project or ViewController project** containing components and logic for the user interface built and written mainly using Java and complementary web languages such as HTML and JavaScript. Both view and controller code are contained in this project.

JDeveloper is optimized to assist you in producing a multi-tier architecture for your database applications. This book concentrates on the power of JDeveloper to produce a multi-tier application centered around ADF BC. A multi-tier architecture encourages the logical separation of the program components.

The following sections describe the ways in which JDeveloper structures and organizes the code needed to support your applications.

## Application Workspaces

An *application workspace* (or just *workspace*) is the highest-level container within JDeveloper. Its contents are projects that contain the code files. A workspace represents all related projects that you need to access in one work session, although you can have many workspaces open at the same time. It corresponds more or less in size and scope to a multi-window, multi-block .fmb file in Oracle Forms Developer. Each application workspace is implemented with a single file that has a .jws extension. This is an XML file that contains the names of the project files (containers for code files) that comprise it.

For example, an application workspace can contain a project for business services (the data sources) and another project for the user interface. The business services project can be used in other workspaces, if needed. When you use business services, the project containing the business service objects must be open in the same workspace as the project that needs to use the business services. You deploy applications a project at a time, but if the workspace contains multiple projects, you will have an option to deploy related projects at the same time.

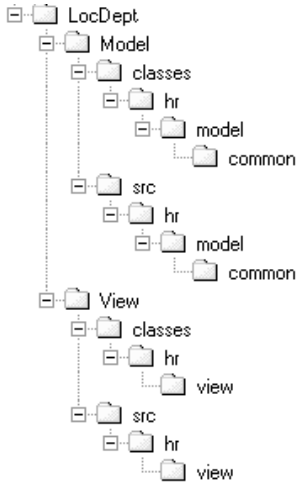
## Projects

A *project* is the next level of container in JDeveloper that contains code files. You can think of a project as a major portion of a program. As mentioned, a program is typically partitioned across two projects: one for the business services (such as ADF Business Components) and one for the UI components. The projects can reside in the same or different application workspace directories, but the business services project must be accessible in the same application workspace as the project that uses it. A project represents a number of code files that are deployed together. Usually, the files within a project are related, for example, the project contains just user interface code for your application. The project is implemented in a single XML file that has a .jpr extension and contains the names of the code files that comprise it.

## JDeveloper Directory Structure

Applications built using JDeveloper will be partitioned across multiple directories. Typically, the application workspace is stored in one directory and the project files within that application workspace are stored in subdirectories underneath the application workspace directory.

As mentioned, within the application workspace directory, JDeveloper normally generates two project files, Model and View, each in its own directory for Java source code and compiled code. Within the Model directory, the source code is contained in the folder labeled "src," and compiled code is contained in the "classes" folder. Within each of these directories, additional subdirectories correspond to any defined packages, as shown here:




Within a business component project's model package directory, an additional folder called "common" stores the generated ADF BC configuration files.

## Development Steps

The general steps you go through to create code in JDeveloper for an application that accesses the database follow:

1. Create an application workspace using a template that contains view and model (business services) projects.

2. Define and test the business services objects.
3. Add code for user interface objects. This code uses objects in the business services project.
4. Test the user interface code with the business services project.
5. Deploy the Model and View projects.

Various practices throughout the book provide specifics about each of these steps and give you experience in creating different styles of application code.

# Hands-on Practice: Build a Client/Server Application Using the JDeveloper Wizards

This practice will give you a feel for how to use the wizards to create a simple client/server–style Java application project. This practice also provides an introduction to working with JDeveloper 10*g* and gives you a basic understanding of its primary components. Although this practice introduces the main application development features of JDeveloper, Chapters 2 and 3 provide details about some of the features introduced in 10*g*. Detailed information about ADF Business Components may be found in Part II of this book. Chapter 7 and Part III contain detailed information about Java applications.

Since the focus in this chapter is to familiarize you with the overall JDeveloper 10*g* architecture, you will only use the higher-level wizards (that produce a complete code module) in this hands-on practice. Normally, you would modify the code created by the higher-level wizards or use the individual element wizards to create the code.

The two projects created in this practice could be partitioned across any number of application workspaces with any number of projects. For this practice, one workspace will be created to hold both projects; this is the default for most applications. Two separate projects will be used: a Model project for the business components and a View project for the Java application. Each project represents a number of files that will be deployed as a unit.

This practice consists of the following phases:

**I. Create the application workspace and database connection**

- Create the application workspace
- Create a database connection

**II. Create the Model project**

- Create ADF BC entity and view objects
- Test the Model project

**III. Create the View project**

- Create the Form

- Generate the JClient user interface components
- Test the application

As with most practices in this book, you will use ADF BC to build the business services (model) objects for the user interface layer.

## I. Create the Application Workspace and Database Connection

If it is the first time you are opening JDeveloper 10*g*, you will see the welcome screen. Otherwise, the product will reopen to show the screen as it was the last time the product was closed.

The IDE window has several work areas as shown in Figure 1-1.

**TIP**

*You can choose to view or hide each work area using the View menu. In addition, you can reposition the work areas in an arrangement different from the default described next. Chapter 2 contains details about how to manage the work areas and windows in the IDE.*

- **The navigator area**, by default, contains tabs for Applications, System, Connections, and Run Manager. If the System and Run Manager tabs are not visible, you may display them using the View menu. The following navigators are available:
  - **The Application Navigator** displays the logical structure of your application including workspaces, projects, and files. Most of your time will be spent manipulating objects in this view.
  - **The System Navigator** displays the physical structure of your application, including workspaces, projects, and files. By expanding all of the nodes, you can view the contents of your projects.
  - **The Connection Navigator** is used to manage access to a database or other external resources.
  - **The Run Manager** navigator is used to display the processes that are running applications you started from JDeveloper. For example, running a web application will cause the Run Manager tab to appear and an entry placed in it for the Embedded OC4J Server.

Chapter 2 contains more information about the navigator windows.

**NOTE**

*As mentioned in Chapter 2, all navigator and other windows can be repositioned from their default location. This chapter describes the locations in which the navigators and other windows appear by default.*

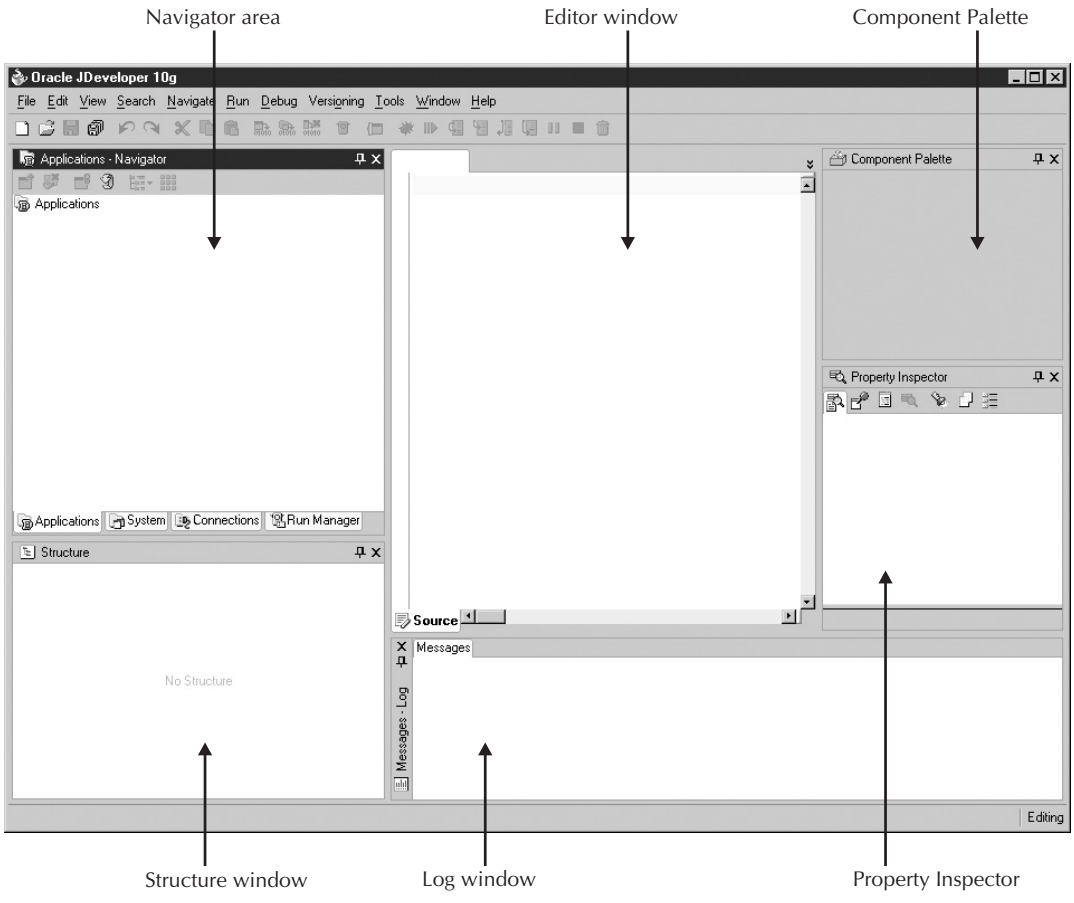


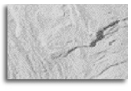

**FIGURE 1-1.** *JDeveloper 10g IDE window*

- **The Structure window** is used to display the contents of the object(s) selected in the Navigator.
- **The Log window** is used to display runtime messages including errors and system processing messages.
- **The editor window** is used to display the editors, visual editors, modelers, and viewers.
- **The Property Inspector** is used to view and change details of the selected object. It is activated when items selected in the navigator have associated properties that can be edited.
- **The Component Palette** is used to add elements to the file in the editor window.

## Create the Application Workspace

By default, JDeveloper places all workspaces within the JDEV_HOME\jdev\mywork subdirectories structure. (As mentioned in the Introduction, this book refers to the directory in which you have installed JDeveloper, such as "C:\JDev10g," as "JDEV_HOME.") Use the following steps to create the application:

1. In the Application Navigator, on the Applications node, select New Application Workspace from the right-click menu.
2. In the Create Application Workspace dialog, change *Application Name* to "LocDept". (See Chapter 6 for a description of Naming Conventions). Note that the last part of the directory name automatically changes to "LocDept" as you type the name.

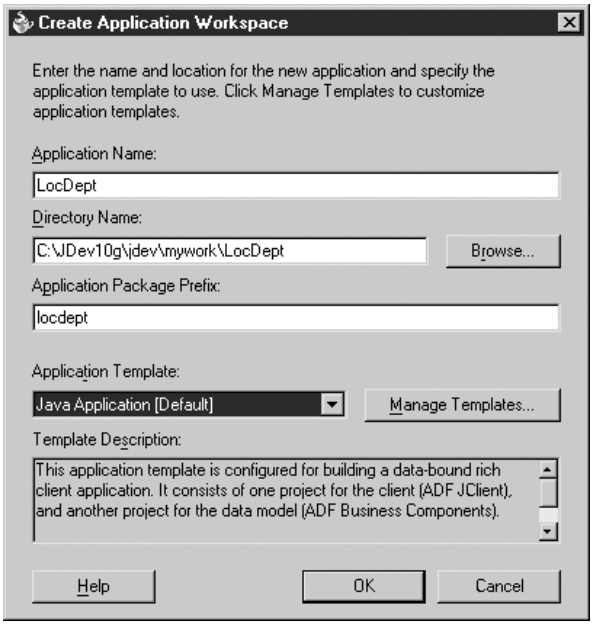


**TIP**
*Dialogs in JDeveloper 10g contain a context-sensitive Help button. Use this button to get information about the fields and options in the dialog.*

3. Enter "locdept" in the *Application Package Prefix* field. This prefix will then be automatically used for creating packages.

   **Additional information:** The *Application Package Prefix* field defaults to the last application package prefix used. If this is the first time you have used JDeveloper, this field will be blank. If not, replace the displayed value.

4. Change the Application Template pulldown to "Java Application [Default]" as shown next. Click OK to dismiss the dialog. Under the LocDept node in the navigator, you will now see Model and View nodes.



**Additional Information:** Different application templates will automatically build different projects for your application as explained before.

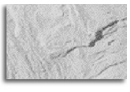
5. Click Save All (in the main toolbar).

## Create a Database Connection

The Connection Navigator contains different types of connections such as Application Server, Database, Designer Workarea, Oracle SCM, SOAP Server, UDDI Registry, and WebDAV Server. More information about these connection types can be found in Chapter 2. For this hands-on practice, you will create a database connection to the HR schema.

The HR schema is one of the demonstration schemas supplied with the Oracle database. Details about setting up the HR database to work with the hands-on practices in this book can be found in the Introduction to this book.

1. Click the Connections tab in the navigator area. On the Database node, select New Database Connection from the right-click menu to access the Create Database Connection Wizard. Click Next if the Welcome page appears.



**NOTE**
*The Welcome page appears by default in many wizards. This page explains the purpose of the wizard and is useful to read when you are learning the product. You can turn off the Welcome page for a particular wizard by checking the "Skip this Page Next Time" checkbox on the Welcome page.*

2. On the Type page, name your connection. Since this practice will use the HR schema supplied with JDeveloper, enter "HR" in the *Connection Name* field. Leave the *Connection Type* as the default, "Oracle (JDBC)." Click Next.
3. On the Authentication page, type "HR" in both the *User Name* and *Password* fields since the practices in this book use the HR schema. Leave the *Role* field blank.
4. Check the *Deploy Password* checkbox. Click Next.
5. On the Connection page, you will need to explicitly set the *Host Name*, *JDBC Port*, and *SID*. Contact your network administrator or DBA if you are unsure of the appropriate settings. Click Next.
6. Click the Test Connection button on the Test Page to check your connection definition. If the settings were correct, you will see a "Success!" message in the status field. If you receive an error message, check the settings on the previous pages by clicking the Back button to return to those pages.

   **Additional Information:** A successful test does not necessarily mean that the database will allow you to build your application successfully. The connection will be successful even if you have no privileges to any table in the system. This test only verifies that you can connect to the specified account.

7. Click Finish. Check that the new connection is listed under the Connections\Database node in the Connection Navigator. The connection has been saved at this point. You do not need to explicitly save it.

   **Additional Information:** You can double click any connection you have created to view, edit, or test it at a later time. Explore the HR connection node. Examine the various objects to see what is contained in the Oracle-supplied HR schema. You also have the ability to drop existing objects or to create PL/SQL program units and database users. JDeveloper includes a simple and convenient utility, the SQL Worksheet, which you can use to enter SQL commands. This may save you from having to use another tool to work with database structures. You can access SQL Worksheet by selecting the Connection Navigator tab, expanding the Database and specific connection nodes. Right click the database connection (in this case HR) and select SQL Worksheet.

**What Just Happened?** You created an application workspace and database connection in preparation for building your JDeveloper application projects. The New Application Workspace dialog and Create Database Connection Wizard are typical of many of the dialogs and wizards you will encounter in JDeveloper.

Notice how the connection and application workspace are completely independent at this point. The workspace is a logical container for building your application and specifies the primary directory where that application will reside. The database connection is stored outside the workspace so that it can be referenced where appropriate. Therefore, any application workspace can use the connection that you just created.

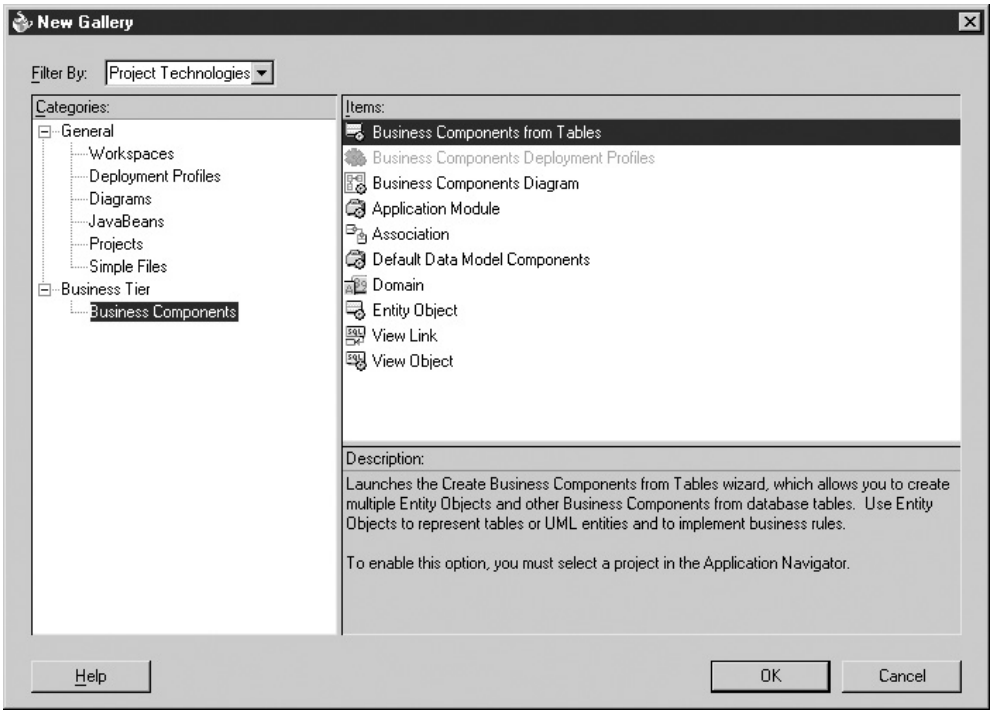
# II. Create the Model Project

Although the HR schema has numerous tables, you will only be defining objects for the tables necessary to make the practice applications run. This represents just one style of application development. An alternate approach would be to build the entity objects to support a large portion, if not all, of the database and then build separate view objects and application modules for each project.

## Create ADF BC Entity and View Objects

Use the following steps to create the Model project:

1. Click the Applications tab of the Navigator area to switch to the Application Navigator.
2. If it is not already expanded, expand the LocDept application workspace node. On the Model node, select New from the right-click menu on the Model node.
3. The New Gallery is displayed. For this practice, leave the Filter By pulldown as "Project Technologies." Select the Business Tier\Business Components category. In the Items list, select Business Components from Tables as shown next.



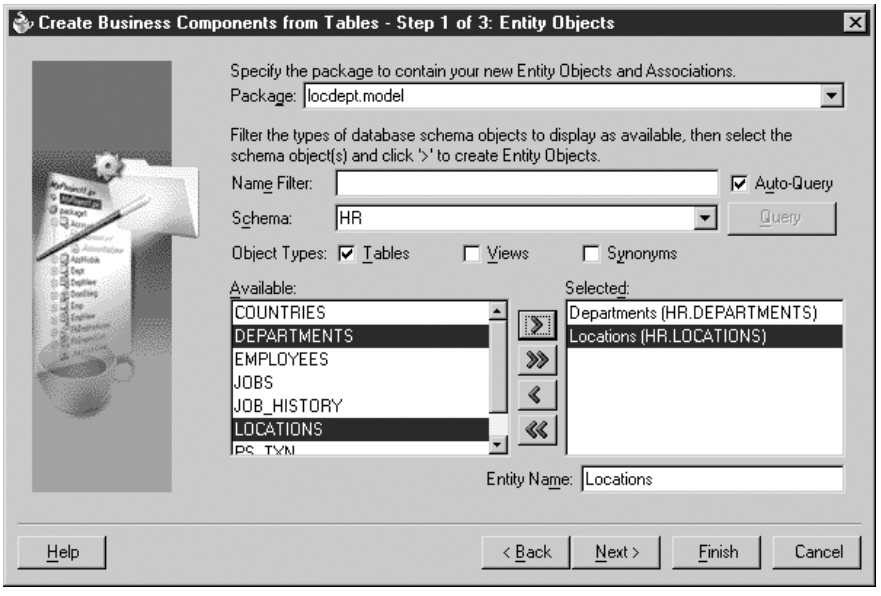

**Additional Information:** The Filter By pulldown at the top of the New Gallery allows you to choose between "Project Technologies" and "All Technologies." The filtered view ("Project Technologies") only displays items that are usually appropriate for the type of project you are creating. To see the entire list of options, select "All Technologies" from the pulldown.

4. Click OK. The Business Components Project Initialization dialog will appear. If you have multiple connections set up, you may not see HR in the *Connection* field. Make sure that HR is displayed. Leave the default *SQL Flavor* and *Type Map* settings. Click OK.

5. Click Next if the Welcome page of the Create Business Components from Tables Wizard appears.

6. On the Entity Objects page, since you specified the locdept application prefix when you created the applications workspace, the *Package* name should show "locdept.model." Ensure that "HR" appears in the *Schema* field. Select DEPARTMENTS. Hold down the

CTRL key and select LOCATIONS in the *Available* pane. Both tables will now be selected. Click the right arrow (">") to move these to the *Selected* pane as shown next.




**Additional Information:** Since there are only a few tables in the HR schema, it is easy to scroll through them to find the one(s) you want. If you have a large number of tables or views on which to base your application, you can filter them using the *Object Types* field (Tables, Views, or Synonyms) and/or by using the *Name Filter* field. This will restrict the available objects list. You can also select different schemas and include database objects from those schemas if the connection you are working under has access to those objects.

**TIP**

*The underscore ("_") and percent sign (%") characters in the "Name Filter" field will work as single- and multi-character wildcard filters, respectively. Each time you change a character in the name filter, the list is re-queried. If the list is long, this can take a long time unless you uncheck the Auto-Query checkbox; the Query button will then be enabled and you can click it to refresh the list.*

7. Click Next. On the View Objects page, the *Package* field will default to "locdept.model." Leave this setting. Use the double arrows to move the entity object names to the *Selected* pane, which will declare view objects for both Departments and Locations. Leave the default *Object Name*. Click Next.

8. The locdept.model package name will already be entered in the *Package* field of the Application Module page. Change the application module *Name* to "LocDeptModule". Leave the *Application Module* checkbox checked. Click Next and Finish to complete the wizard.

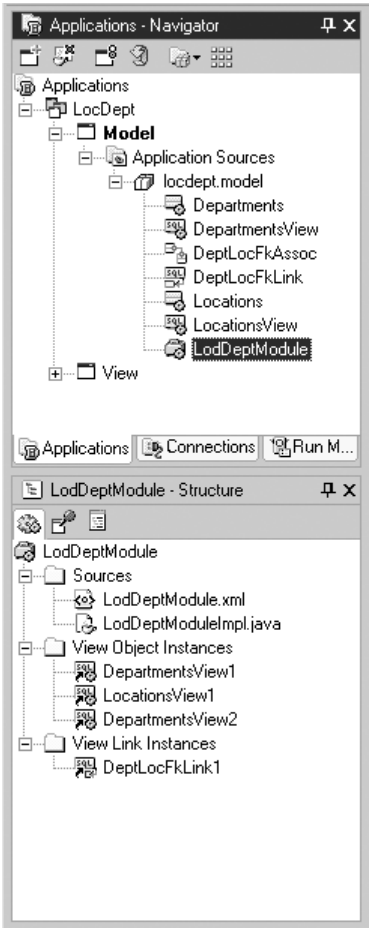
**NOTE**
*A "package" is a logical container for a number of class files. It is represented in the file system by a single directory. Java code references this package name in the class file and in fully qualified names for a method. Chapter 6 contains more information about packages and how they are named.*

9. Click Save All.

## Test the Model Project

You now have ADF BC objects that can be used to connect the user interface to the database. This section shows how to test the ADF BC objects without creating a user interface project. It is useful to know that the model project objects work before creating the user interface code.

1. You will see the new entity objects and view objects under the Model node of the Application Navigator as shown here. On the LocDeptModule node of the Model project, select Test from the right-click menu.



2. Click Connect on the Oracle Business Component Browser - Connect dialog. The Oracle Business Component Browser will open as shown in Figure 1-2. The Log window will display any error messages when the code is compiled.
3. On the LocationsView1 node, select Show from the right-click menu. Scroll through the records using the blue navigation arrows to test the database query.
4. If the Run Manager is not displayed in the navigator area, select Run Manager from the View menu. This will add a Run Manager tab to the navigator area. You can use this window to check on what processes are running.
5. Click the Run Manager tab. Select Terminate from the right-click menu on the Processes\ locdept.model.LocDeptModule node. This will close the Oracle Business Component Browser. This is one way to close the browser. You can also click the "X" icon in the top-right corner of the browser window or select **File | Exit**.

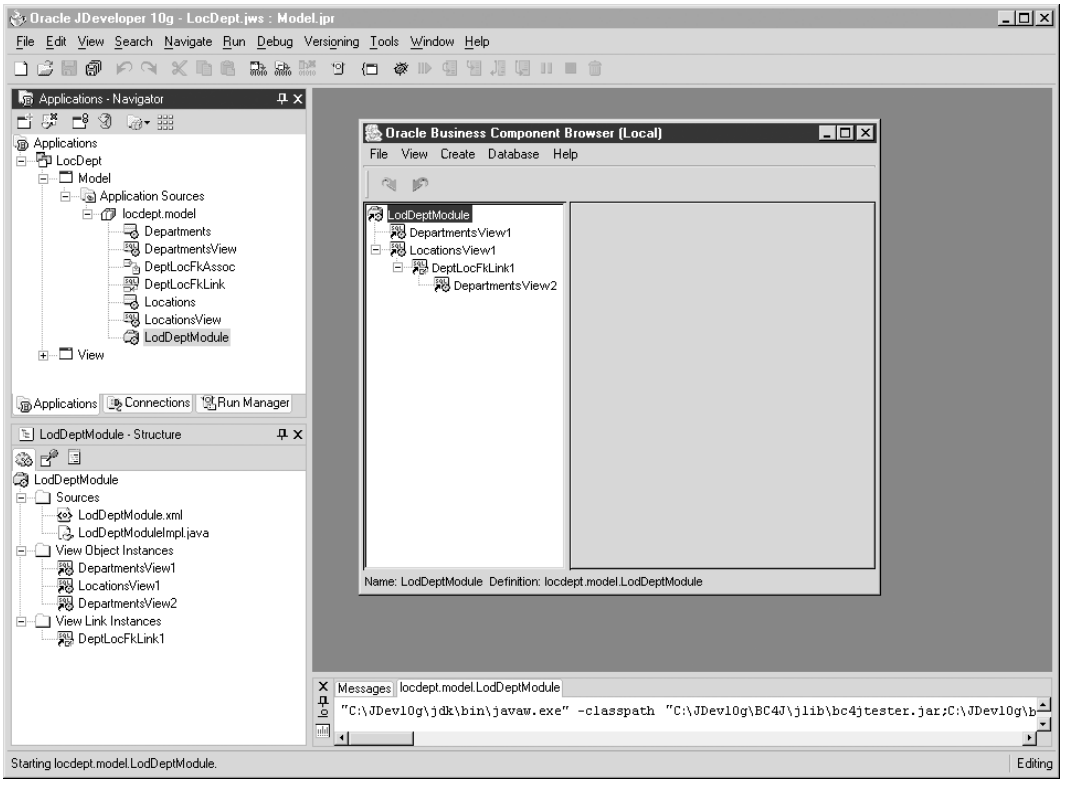



**FIGURE 1-2.** *Oracle Business Component Browser*

6. Click the Applications tab to return to the Application Navigator.
7. Click Save All to save your work.

**What Just Happened?** In this phase, you made a few simple selections and allowed the JDeveloper wizards to create an application project including information from the DEPARTMENTS and LOCATIONS tables of the HR schema. Using the high-level wizards, you created ADF business component objects that are exact images of the underlying DEPARTMENTS and LOCATIONS tables of the HR schema.

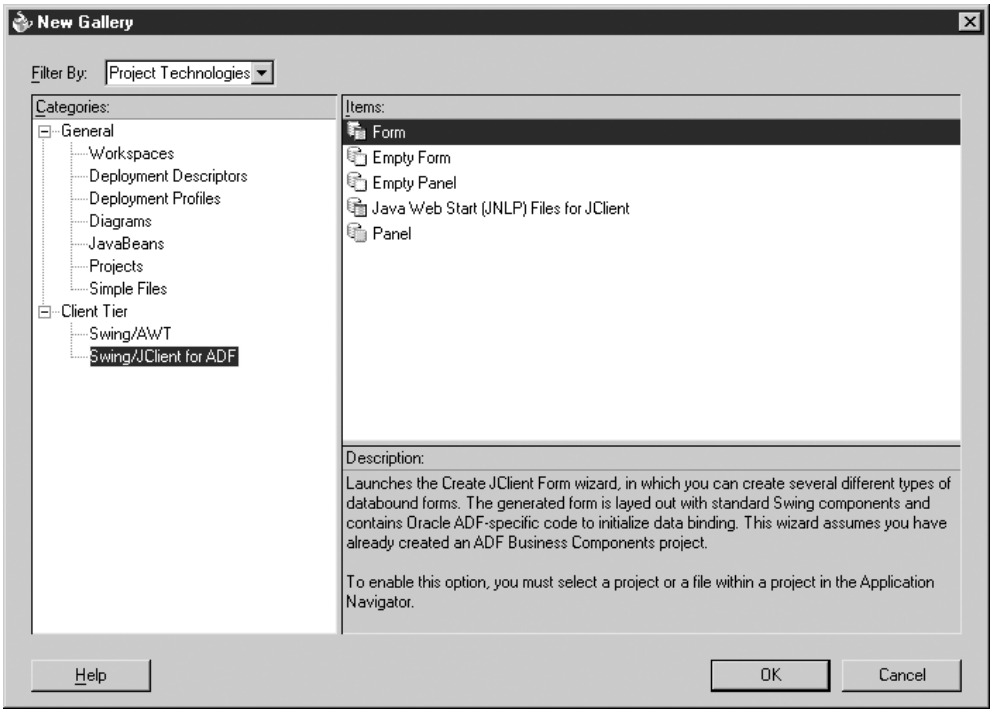
# III. Create the View Project

In this phase, you will create a Java application form (a particular style of Java program) and link it to the Model project in the LocDept workspace created earlier. For more information about Java applications, see Chapter 7 and Part III of this book.

## Create the Form

Use the following steps to create a simple Java application form:

1. In the Application Navigator on the View node, select New from the right-click menu.
2. In the New Gallery, under the Client Tier node, select Swing/JClient for ADF in the Categories list and Form in the Items list as shown here:



3. Click OK to start the Create JClient Form Wizard. Click Next to dismiss the Welcome page if it appears.
4. On the Form Types page, select the "Master-Detail Tables" radio group selection. Leave the default selection of "Form" for the implementation radio group. Click Next.
5. On the Form Layout page, leave the default for *Select a Master template* ("Single columns (label left)"). Change the *Number of columns* to "2." Leave the *Select a Detail template* set as "Table." Click Next.
6. To attach the View project to the Model project, on the Data Model page, click New to start the ADF Business Components Client Data Model Definition Wizard. Click Next to dismiss the Welcome page if it appears.
7. Since there is only one business components project created, leave the default settings for the Definition page. Click Next and Finish to complete the Client Data Model Definition Wizard.

### Generate the JClient User Interface Components

These steps will create the files and Swing components to display the data in the application.

1. On the Data Model page of the JClient Form Wizard, the name of the model you just created (LocDeptModuleDataControl) will be displayed in the "*Select the Data Model Definition*" field. Click Next.
2. Leave the default settings on the Panel View page and click Next.
3. By default, all of the attributes will be selected on the Attribute Selection page. Click Next. Click Next again on the Attribute Selection page for detail view object attribute selections.
4. On the File Names page, the *Package name* "locdept.view" should already be filled in. Leave the other default settings. Click Next and Finish. You will now see a number of .java and .xml files under the View node in the Application Navigator and other window areas will become active as shown in Figure 1-3.

   **Additional Information:** You can now examine the Structure window to navigate the layout objects of the form. The Component Palette also opens to provide options for adding functionality and the Property Inspector for the PanelLocationsView1UIModel is displayed.
5. Click Save All.

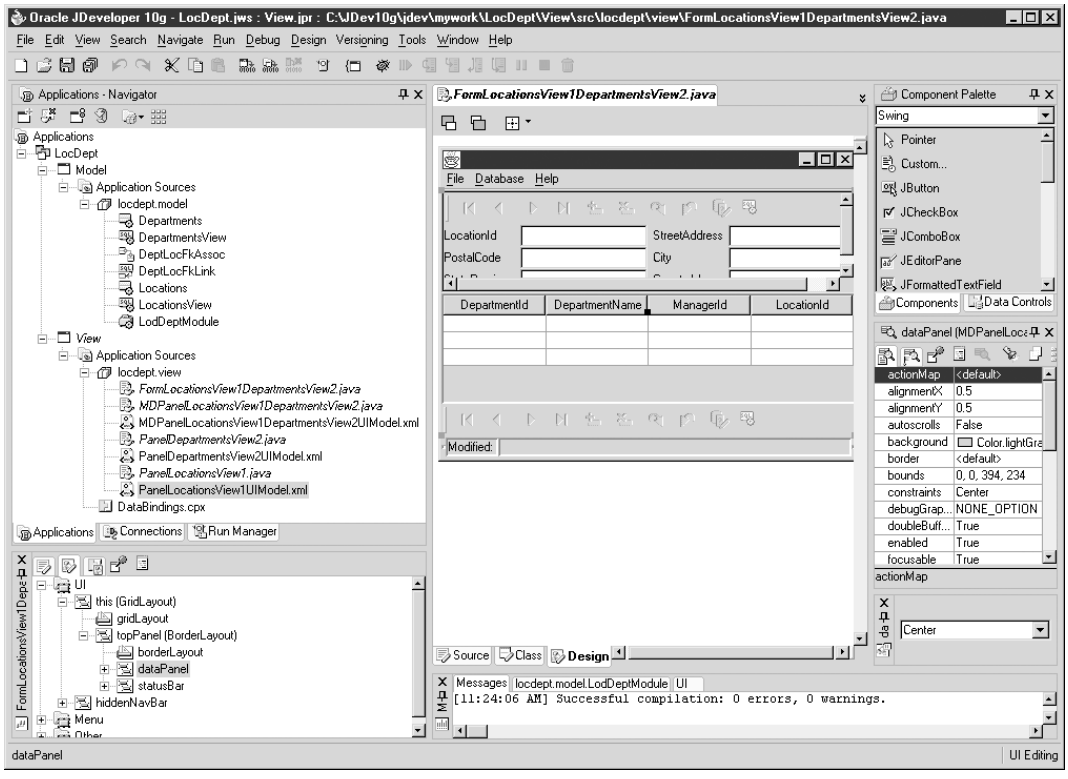



**FIGURE 1-3.** *Form created by JDeveloper wizards*

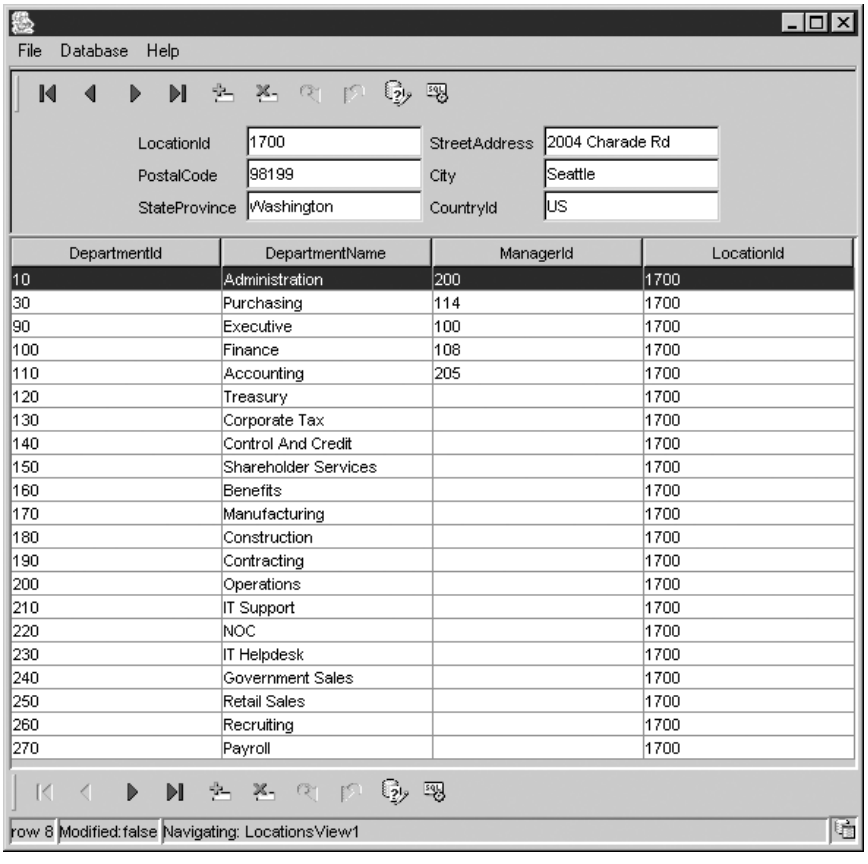
## Test the Application

To ensure that the application is working at this point, you should test it using the following steps.

1. On the FormLocationsView1DepartmentsView2.java node under the LocDept\View\Application Sources\locdept.view node in the Navigator, select Run from the right-click

menu. Alternatively, you can click Run from the main toolbar to run the form shown here.

**TIP**
*Location 1700 has many departments.*




2. Scroll through the records to see the data. Notice that the departments shown in the detail table are the details for the displayed location in the master area above. Close the form using the "X" icon in its upper-right corner or use **File | Exit**.

**What Just Happened?** In this phase, you created a simple Java application. You allowed the JDeveloper wizards to create all of the code and viewed the results of using the default settings for creating the application.

The first thing you did was to define an application workspace containing two projects for Model and View. You then set up the Model project using ADF BC components. You then ran the wizard to create a default user interface application. This is a fully functional application that you can use to select, insert, update, and delete records. It is not a production-level application,

but if you carefully review the generated objects, you can use this code to get an idea of how you should structure a Java application that interacts with ADF BC.

To summarize the process of creating a Java application using the wizards, you would use these steps:

1. Create the application workspace with Model and View projects.
2. Create an image of the database using entity objects and associations.
3. Create a client data model for binding the ADF BC application module to the user interface.
4. Create the user interface application.

# Hands-on Practice: Create a Simple JSP Page

This hands-on practice demonstrates how to create a simple JSP page to use a web browser such as Internet Explorer or Netscape to browse the Locations table of the HR schema. This project uses the same ADF BC project that you created for the earlier hands-on practice. JDeveloper 10*g* includes an easy-to-use visual editor with standard word processor editing capabilities and a drag-and-drop interface. For additional information about JSP technology, see Chapter 18.

This practice consists of the following phases:

**I. Create the JSP project**

**II. Create the JSP file**

## I. Create the JSP Project

This project will reuse the ADF BC project built in the preceding practice and build an additional JSP project for an alternate user interface. If you have not completed the first practice in this chapter, complete through phase II before beginning this phase.

1. If not already displayed, click the Applications tab in the Application Navigator. Select New Project from the right-click menu on the LocDept node.
2. In the New Gallery, the Projects node will already be selected in the Categories list. Select Web Project from the Items list. Click OK.
3. If the Welcome page of the Create Web Project Wizard is displayed, click Next to display the Location page. Change the project name to "ViewController". Click Next.
4. On the Web Project Profile page, leave the default settings in all of the fields and check the *Add JSP Page* checkbox. Click Next and Finish.
5. The Create JSP dialog will open. Change the *File Name* to "Loc.jsp". Click OK.
6. Click Save All.

   **Additional Information:** You will now see a blank editor window for the JSP file in the center of the screen, the Structure window below the Application Navigator; and the Component Palette and Property Inspector on the right side of the screen.

**What Just Happened?** You just created a blank JSP project and an empty JSP file.

# II. Create the JSP File

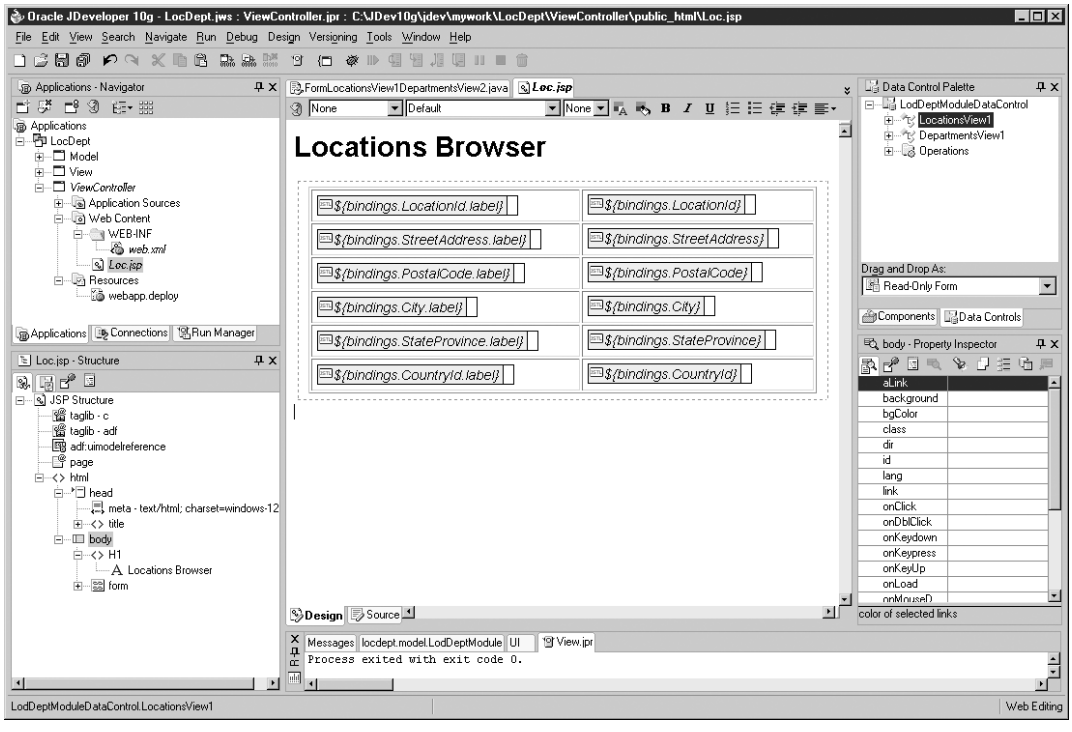
You now need to create the JavaServer Pages user interface components for the application, using the JSP/HTML Visual Editor. In terms of code, JSP files consist of HTML (or other language used to control page display and formatting) and Java tags (which supply data and processing). In JDeveloper, you will format the page and then add JDeveloper components to the layout.

1. Click at the top left of the blank editor page. Type "Locations Browser" as the JSP Page heading.

   **Additional Information:** Note that as you type, the words are added to the Structure window under the body node.

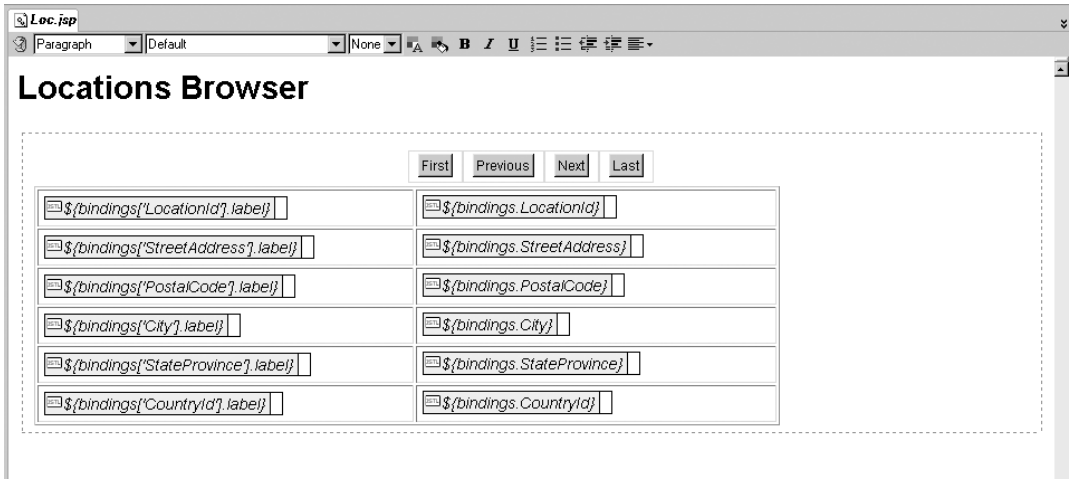
2. Select Heading 1 from the first (Block Format) pulldown. This will change the font of the line and add the appropriate HTML tag to the file. Press ENTER to move the cursor to the second line.

3. Select the Data Controls tab. In the Data Control Palette, select LocationsView1 under LocDeptModuleData Control. Select "Read-Only Form" from the *Drag and Drop As* pulldown and drag LocationsView1 onto the editor under Locations Browser. Your screen should look something like this:



**Additional Information:** You can format a JSP file using HTML or other page formatting options. The Component Palette changes to show HTML components, but you could use cascading style sheets, JavaScript, or other technologies that are compatible with JavaServer Pages and HTML coding. You can also use the text editing features of this window to modify the font, type, size, style (bold, italic, underline), orientation (left, center, right), and so on.

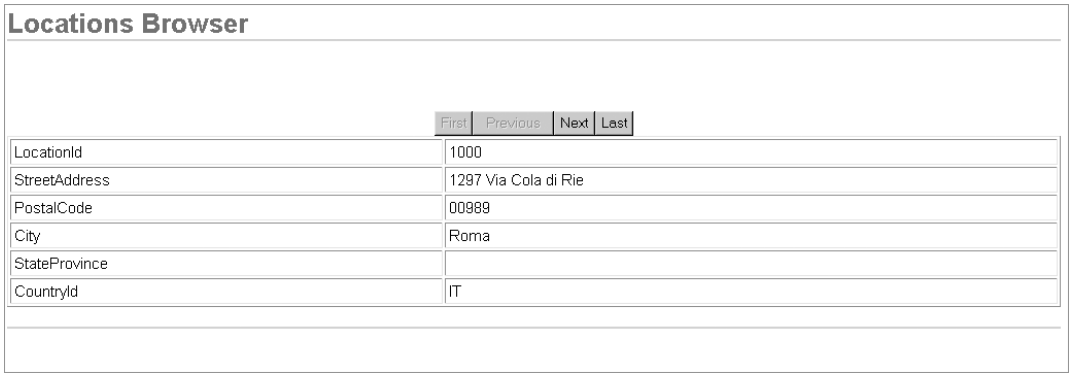
5. Click above the top-left corner of the form inside the red dotted line. Press ENTER to create a blank line.
6. Select Navigation Buttons in the Drag and Drop As pulldown. Drag LocationsView1 to the cursor location.
7. Center the buttons using the alignment pulldown on the JSP page formatting toolbar and selecting Center or select "center" in the pulldown of the *align* property in the table Property Inspector. The visual design should look like the following:




**Additional Information:** The alignment pull-down is on the far right of the formatting bar at the top of the design window. If it is not visible, expand the width of the window.

8. Apply a template to set the look and feel of your JSP page by selecting the Components tab. Select CSS from the pulldown.
9. Click JDeveloper in the CSS pulldown (you do not need to drag and drop this item) and note the changes in the editor window.
10. Click Save All.

11. Test the JSP by selecting Loc.jsp in the Navigator and clicking Run from the right-click menu. This will start the Embedded OC4J Server and open your default browser. The browser will display the Locations data as shown in the following portion of the browser screen:




12. Test the navigation buttons to browse the Locations table. Close the browser window.

**What Just Happened?** In this phase, you created a simple JavaServer Pages file. This demonstrates the power and flexibility of JDeveloper, which can be used for both Java applications and web pages. The new visual editor provides an easy-to-use interface that allows you to create more attractive JSP pages with foreground and background color, font, and text style options that were not possible in earlier versions of JDeveloper.

The authors suggest that you carefully follow the practice steps and examine the generated structures and code. Understanding how JDeveloper generates application elements will enhance your understanding of the rest of the material in this book.