CREATING JAVA APPLICATIONS, APPLETS, AND JSPS IN JDEVELOPER

Peter Koletzke, Quovera

Toto, I have a feeling we're not in Kansas anymore. We must be over the rainbow.

—Dorothy, The Wizard of Oz, MGM (1939)

If we apply Dorothy's observations to the world of Java development, truer words have not been spoken. We no longer inhabit the familiar world of PL/SQL where we understand and are in tune with the methods for developing and deploying applications. There are many promises over the rainbow, but to realize them, we need to become familiar with new development and deployment methods.

JDeveloper offers a powerful interface for quick development of bug-free Java code. Since it is Oracle's premiere Java tool, you can use it to create all different styles of applications that the Java language supports. The challenges are in understanding the different types of applications and in knowing where to go in the tool to get started.

This paper explains the characteristics of three main web deployment options that you can create using Java: Java application, applet, and JavaServer Page (JSP). It briefly explores the architectures of each of these options and compares their benefits and drawbacks. The paper also discusses how to get started developing each alternative using the JDeveloper wizards. The JDeveloper help system contains demo applications and tutorials that will step you through the process of creating the code.

Since you will have to complete the default applications that the JDeveloper wizards create, the paper will also guide you in the steps that are required to create a production-ready application inside and outside of the wizards. The actual deployment process is easily accomplished using the Deployment Profile Wizard (although details of this step are outside of this paper's scope). Since all deployment options use Oracle's Business Components for Java framework, it is useful to start by briefly explaining this feature. Further explanation is outside of the paper's scope but is available in the JDeveloper help system.

Business Components for Java

Oracle's database abstraction layer, called *Business Components for Java* (BC4J), offloads much responsibility from the application programmer. It provides a set of class files that you can use to easily hook into database objects such as tables and views. Before BC4J, you had to write low-level JDBC (Java Database Connectivity) code that allowed your Java code to access database objects. BC4J hides these low-level calls in a more abstract layer and allows you to concentrate on the application logic instead of Java database access mechanics. Creating a BC4J layer in JDeveloper is easily accomplished by means of running the Business Components Project Wizard. JDeveloper also helps in creating a deployment package that contains the many class files that this wizard creates. A single BC4J layer can service many applications and is not specific to a particular deployment style.

Common Development Steps

The steps that you use to create an application using JDeveloper are slightly different depending on the style of application that you are developing. The following are the common steps that you would follow in JDeveloper regardless of the deployment method.

1. **Create a workspace.** The top level of file organization in JDeveloper is the workspace. It is really just a logical container for a number of project files. Project files contain the actual program code files (such as .java, .html, and .xml) and are used for creating deployment packages. The workspace may include projects that are also used by other workspaces.
2. **Define the BC4J project.** This may be an existing BC4J project that you just include in the current workspace. The BC4J project stores the objects that will allow your program to access the database. After defining this project, you add code and definitions that implement specific business rules.

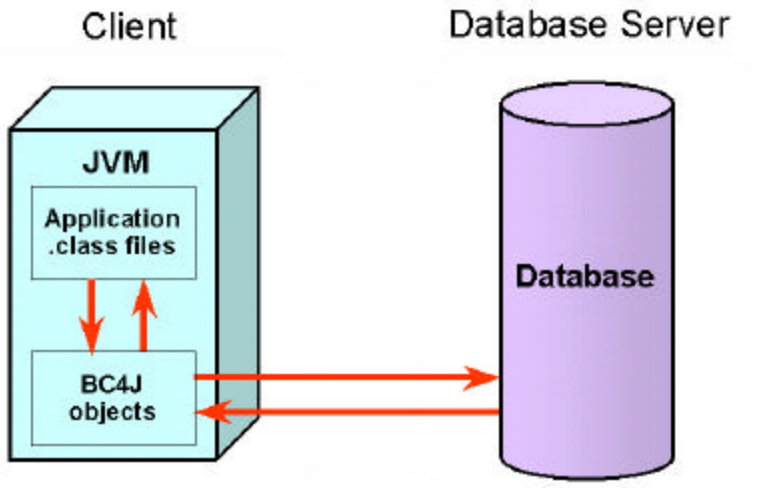
3. **Test the BC4J layer.** You can use the internal Oracle Business Components Browser to test the BC4J objects and your additional code. This is a recommended step because you can verify that all code in this layer is correct before building the user interface code that uses this layer.
4. **Run the Project Wizard for the application.** The Project Wizard creates the project file for the application and automatically runs the specific wizard for the style of application you will create (such as an applet).
5. **Run other wizards as required.** Use the object gallery (**File→New**), to add files to the project for the user interface. All wizards create a certain amount of code that you will need to modify to fit your specific application requirements.
6. **Modify the generated code.** You will need to work other application-specific features into the generated code or modify the default behavior by writing code. Depending on the wizard that you ran, the effort you need to put into this will vary. Some wizards create very complete code whereas others create an emptier shell that you have to fill in significantly.
7. **Test and deploy the code.** You need to perform unit tests of the code within the JDeveloper environment first. This allows you to use the native JDeveloper debugger to resolve problems. When the test and fix cycle is complete, you create a deployment package using the Deployment Profile Wizard and move this to the final test platform that more closely mirrors the actual runtime environment. For example, if you are creating an applet, you would package the required class files and move them with the supporting libraries to a web server. This would allow you to test the deployment package as well as to check that the code works properly outside the JDeveloper environment.

These are the steps that are common to the various application deployment styles. The following discussion explains these styles and gives an idea of how these common development steps would be different.

Java Applications

The term “Java application” refers to a particular style of Java code. In Java terms, the code is really just an ”application,” but since that term is a common one in the IT world, it is usually preceded by the word “Java.” A Java application runs on the client machine in a Java Virtual Machine (JVM) runtime process. The source code .java files are compiled into bytecode (.class) files and stored on the client, or on a local or wide area network server. There is no web server required and the runtime environment is located on the client outside of a browser. Therefore, a Java application runs in a typical client/server environment.

If the Java application uses BC4J, the BC4J objects may be located on the client side as well. As shown in Figure 1, the application's class files access the BC4J objects (class and XML files). The BC4J objects provide the data communication link



to the database. This is called a “*local*” database connection because the database layer is local to the client machine.

Figure 1. Java application architecture - method 1

It is also possible to locate the BC4J objects on a server that runs an application for serving code. An example would be a Common Object Request Broker Architecture (CORBA) server or an Inprise VisiBroker wrapper that connects the application to the BC4J application module located somewhere other than the client machine. This variation, depicted in Figure 2, allows clients to share the main connection and database layers. There are a few other variations on this architecture based on where the BC4J code is located.

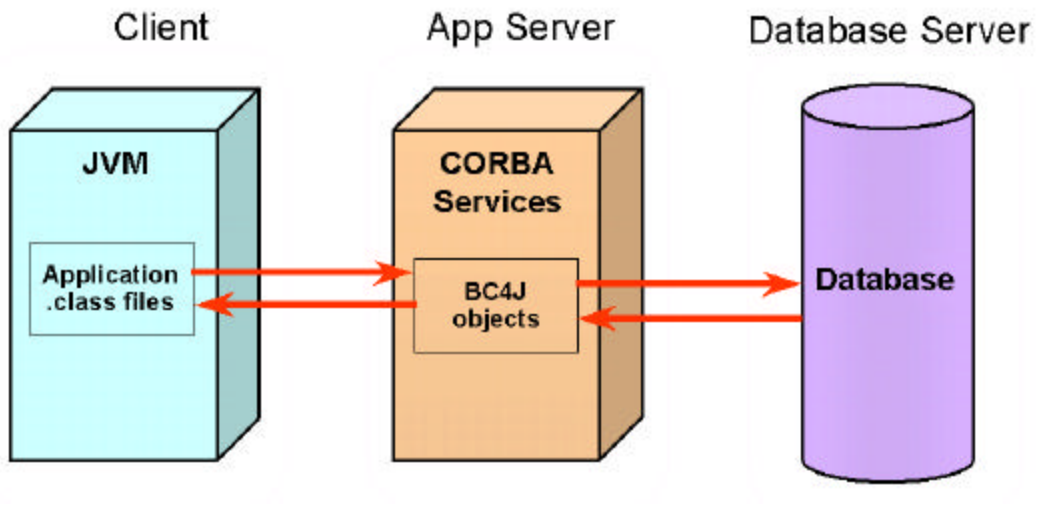


Figure 2. Java application architecture - method 2

Java applications must have a method called `main()`. This method usually calls a constructor method (in JDeveloper this method is `jbInit()`). The constructor method creates the first object, such as a frame. All requests for data flow from the application's frame through the BC4J layer.

To deploy a Java application, you install java.exe (the Java runtime JVM) and supporting Java libraries on the client. You also install the .class files for the application and data access (BC4J) objects and set up the client's CLASSPATH so that the JVM can find these .class files. To run the application, the user enters the following at the command line (or in a shortcut icon):

```
java DeptEmpApp
```

In this example, "DeptEmpApp" is the compiled class file that contains the `main()` method that starts the application. The client machine would likely use a shortcut icon instead of requiring the user to type a command-line string. Figure 3 summarizes the startup sequence from the command line to running the Java application.

When to Use Java Applications

Java applications are indicated for intranet or small-department solutions with a small number of clients. When the number of clients grows, you will experience all of the same problems and resource drains as in client/server applications because, for each new client, a new software installation is required. If you do not want to worry about browser limitations and firewall restrictions and are able to easily manage local client-side installations, the Java application is the proper style.

Advantages of Java Applications

If you are accustomed to client/server deployments, Java applications provide you with an easy architecture in which to deploy Java code. The user interface responds to user events quickly because the code is running on the client. You gain all of the benefits of the Java language, such as object-orientation and portability, without the need to learn and configure the application server.

GUI Controls

Java applications provide rich GUI possibilities. The available libraries of GUI controls (mainly AWT—Abstract Windowing Toolkit—and Swing) provide all of the functionality of traditional windowed applications but allow you the flexibility of modifying each aspect of the control. In addition, Oracle supplies data-aware controls (DAC) (called InfoSwing controls in the JDeveloper component palette) that are based on the feature-rich Swing controls and connect quickly to the BC4J objects

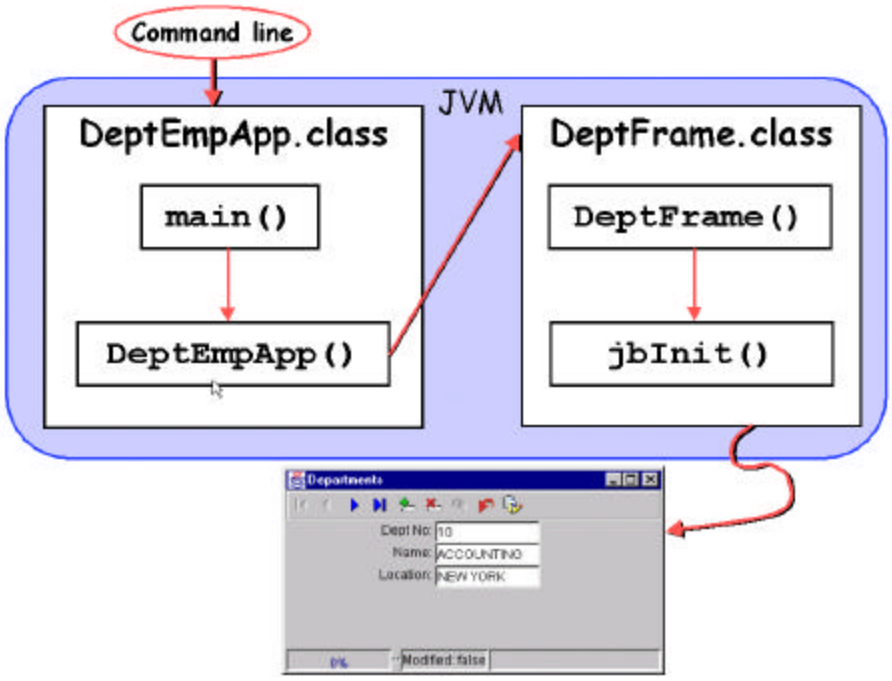


Figure 3. Java application startup sequence

using a single property (*dataItemName*). This makes connecting a Java control in the user interface to a data element in the database a simple task. It is just a matter of laying out the component in the UI Designer and specifying the data source for that component in the Inspector window. There is an additional layer involved with this mechanism that is implemented through controls in the InfoProducers section of the JDeveloper component palette. These controls are non-visual data hooks to the BC4J layer. When you connect a DAC component such as a text field to the BC4J layer, you do so by specifying the details of the InfoProducer component in the *dataItemName* property. Figure 4 illustrates how this connection is made. The InfoProducer components (row set and attributes) act as an intermediary between the GUI controls and the BC4J layer. There is an additional element, called a *connection*, which identifies properties of the database in which the objects are located.

Layout Managers

In addition to the Swing, DAC, and InfoProducer components, you can also use a Java feature called *layout managers* to manipulate components at run time. A layout manager is an object that you define and attach to a container (such as a panel) using the *layout* property. It is responsible for resizing and repositioning the components inside that container when the user resizes the outer window. This is useful because you can deploy the Java application on diverse platforms and be assured that the layout manager will maintain your design regardless of differences in the hardware or JVM used for display.

There are many layout managers and each has a different behavior. For example, the FlowLayout manager is responsible for wrapping components to the next line if the window width is narrowed. You define the outer container (usually a panel) with the FlowLayout manager and it takes care of this wrapping feature automatically. Another layout manager, GridLayout, is can used for a grid effect (such as a calendar) where all areas are maintained at the same width and height regardless of the outer container's size. Layout managers are a strength of Java applications because you can take advantages of them in your code with minimal effort.

Disadvantages of Java Applications

Java applications also come with all of the drawbacks of client/server architecture. One main problem is that runtime and application code must be maintained and installed on the client. WAN servers promise to ease the burden, but the reality is that they are often not responsive enough, so companies use LANs instead. The LAN solution for a large application is still

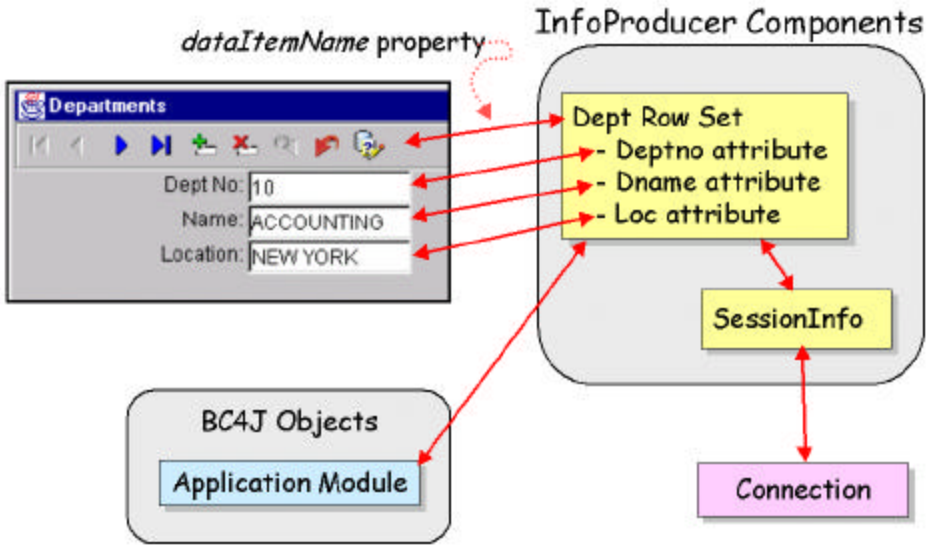


Figure 4. DAC application data communication lines

not responsive and requires installations of the same code on more than one machine. This takes a lot of time and effort, as those who support client/server applications have experienced. In addition, the client needs a large amount of resources because the application is running in its memory and using its disk space.

When the number of users grows, this architecture scales poorly. More users may require additional installations and further decentralization of the runtime code. The architecture makes no use of the benefits of web server technology for centralized installation and maintenance, although the CORBA server architecture mitigates this effect somewhat.

Creating Java Applications in JDeveloper

JDeveloper assists in helping you create the code files for Java applications. There are two different methods that you can use to create an application using the BC4J layer as follows:

- Use the Business Components Data Form Wizard
- Use the Application Wizard.

Both methods assume that you have created a connection object that specifies the proper schema and database attributes. You also must create or open a BC4J project that contains the database objects required for the application. The assumption here is that you have written business rules as much as possible into the BC4J layer and do not have to supplement that after laying out the user interface.

Business Components Data Form Wizard

This wizard starts automatically when you select "A project containing Business Components" as the project type in the Project Wizard (**File→New Project** from the menu). After the Project Wizard completes the task of creating the project file, it launches the Business Components Data Form Wizard. There are other methods for starting this wizard such as selecting **File→New** and double clicking the Business Components Data Form icon on the Objects tab.

The Business Components Data Form Wizard will step you through the various features of the application using a standard wizard interface. When the wizard completes, you will have a complete working application. It will have a single-table or master-detail layout using grid or single-record displays. The proper data connections and layout managers will be assigned. Generating several pages of working code in this way will take you a matter of minutes.

Application Wizard

Another method for creating a Java application is to use the Application Wizard. This wizard starts automatically when you select "Application" as the project type in the Project Wizard. There are a few other ways of starting the Application Wizard (such as selecting **File→New** and double clicking Application in the Objects tab).

The Application Wizard is a single page dialog where you specify the frame that you want to use (existing or new) and whether you want to create a Business Components data form (using this option will launch the Business Components Data Form Wizard described above). If you choose not to create this data form, you can specify an empty frame and the wizard will call the Frame Wizard, which is responsible for naming the frame class and creating the frame object. You then need to create the layout manually using the following steps:

1. **Add InfoProducer objects.** As mentioned above, the row set object and attributes are required to implement a link from the BC4J layer to the application's GUI controls. The row set corresponds to a cursor and the attributes correspond to the individual columns that are retrieved from the cursor. INSERT, UPDATE, and DELETE actions occur through the row set as well.
2. **Add containers and assign layout managers.** The components that you add to the user interface must be placed inside container elements such as panels. Each panel must be assigned a layout manager that is responsible for its behavior when the container is resized. Usually, you create a set of containers inside of containers based on the requirements of the interface. Designing the container hierarchy requires knowledge about the layout managers and about the design of the application. At this stage, you also assign the properties of the layout manager such as the gap between horizontal areas.
3. **Add the user interface (UI) components.** You add controls that the user interacts with, such as text fields and buttons manually. It is important to place them in the proper container based on your design. One technique that Java programmers use is to initially lay out the components using the XYLayout manager. This layout manager imposes no repositioning or resizing behavior on its components so you do not have to worry about those effects at this point in the process. After all components are in the proper place, you can create additional containers as required and substitute the desired layout manager.
4. **Set the component constraints property.** The components are positioned within the layout manager using a property called *constraints*. For example, the BorderLayout manager allows you to define five areas within the container (East, Central, West, North, and South). To assign a particular component to a specific area, you set the component *constraints* property (for example, to "North"). Most layout managers only allow one component per area so you may have to embed additional containers for additional components. Most layout managers do not allow you to set a specific X and Y position for a component because the layout manager is responsible for assigning those based on the panel's shape and size. Therefore, performing precise layout in the UI Designer in JDeveloper is not as important as assigning the layout manager and *constraints* properties.
5. **Set the component properties.** Properties of the components, such as size, font, and label are manipulated on the Inspector window. If there is a property of the object that does not appear in the Inspector, you can write `set()` code for the property in the Source Code Editor. As mentioned, the size and position properties of the component may be partially affected by the layout manager of the container in which it resides.
6. **Add event code.** The Inspector also allows you to create events for objects. For buttons and other event components, you can double click the event name (such as *actionPerformed*) in the Inspector to create the listener code and event stub code. You then need to fill in the stub with code that is specific to your need. Code that you would add might conditionally navigate to other forms or applications or to process a mouse click.

Which Java Application Method is Best?

This is an important question. The core of the answer is common sense: if the application that you need to create is close to what the Business Components Data Form Wizard will create, this is the wizard to use. It is virtually guaranteed that the wizard will not fulfill 100% of the application's requirements. Therefore, you will need to dig into its code and supplement or replace the defaults. This requires an understanding of how the generated code is built and structured. If the final application requires significant modification of the generated code, it is probably better to start from scratch without using the Business Components Data Form Wizard.

> **Note**
>
> Because of JDeveloper's open architecture, another alternative for development of any of the methods mentioned in this paper is to create your own wizard. You can modify an existing wizard or create one for a particular style of application. This requires substantial amount of effort and a high level of knowledge of the Java language. There is a published API (the *Addin API*) and some examples provided by Oracle, but you will only find a payback for the effort if you will be creating many applications of the same style..

Applets

Applets are an alternative that better leverages the strengths of web technology. With a Java applet, when the application is first run, the applet is copied from the application server to the client machine and run within a browser session. In subsequent executions of the same version of the applet, a copy of the applet located on the client machine is run. Applets give you all of the functionality of a Java application as if you were deploying client/server but allows you to maintain it over the Internet. As with Java applications, you are working entirely in a Java environment and can use JDeveloper to create the code.

As with Java applications, applets allow you to use the rich user interface components that are offered by AWT and Swing libraries. The applet differs from the Java application only in the way it is started (from a browser) and in the initial location of the code, which is an application web server. The steps in the applet startup process follow:

1. **The client browser requests an HTML file** from the web server through a standard URL. The HTML file may be static or dynamically generated from another application. This HTML file contains a special applet tag such as the following**:**

```
<APPLET CODE = "empappjsp.DeptApplet"
   CODEBASE = "/applet_code/"
   WIDTH = 400
   HEIGHT = 400
   ALIGN = middle >
</APPLET>
```

2. **The applet tag signals the browser to start an applet window** for the JVM session and load the application's class file named by the *code attribute.* The applet tag's attribute *CODEBASE* specifies the location of the applet's .class files relative to the physical location of the HTML file. If the HTML file is in the same directory, the CODEBASE attribute is set to "." (the same directory). The CODE attribute finds a specific class file in a directory listed in the CLASSPATH variable on the application server.
3. **The application's .class files download** (the first time) from the application server and are presented in the applet window. The class files will be cached (in Java version 1.3) on the PC's hard drive for the next time the applet is run.
4. **The applet class file `init()` method that calls the `jbInit()`** method to start the applet.
5. **The code runs within the PC's JVM** the same as a Java application. The database communication occurs between the JVM and the BC4J objects as with the Java application.

Figure 5 shows the communication lines for the applet architecture. As with the Java application, the BC4J files may alternatively be located on the client. Figure 6 depicts the calling sequence for an applet.

When to Use Applets

Applets are indicated for use within an organization. An applet gives you a client/server-style application without the overhead of maintaining a client/server environment. An intranet environment may also provide adequate bandwidth to provide a workable initial load time for large Java applets. Since an intranet system is a controlled environment, you will likely be operating behind the firewall, thus eliminating the security restrictions.

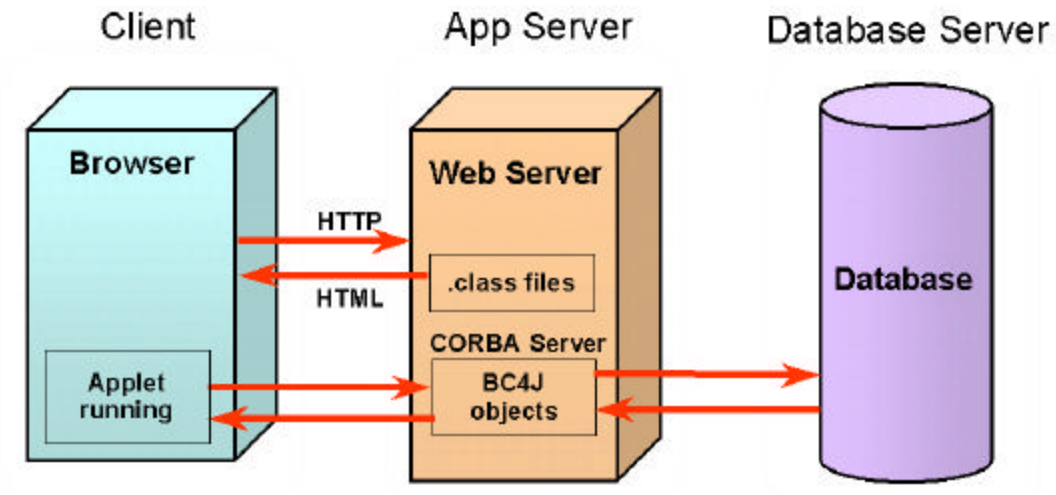


Figure 5. Applet architecture

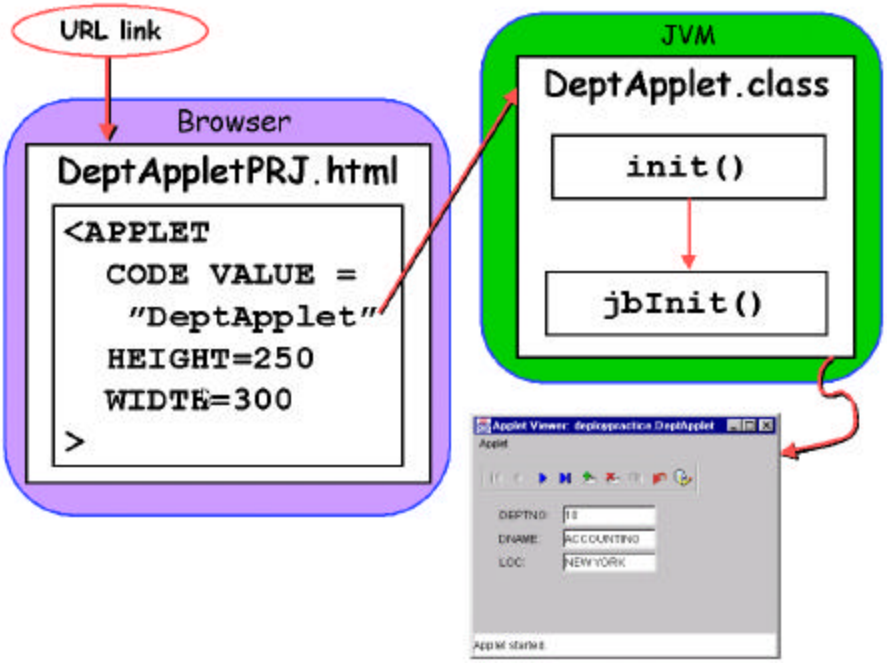


Figure 6. Applet startup sequence

Advantages of Applets

For the most part, the same advantages that apply to Java applications apply equally to applets. An applet has the additional advantage of allowing you to use the web application server to store a common set of runtime files. Although the client still needs to have the JVM runtime files installed, they are included with all popular browsers. This means that the burden of installing software on the client is greatly reduced. The main installation requirement is the client's browser, which is standard issue these days.

Disadvantages of Applets

Java applets can be very large so the initial download time may be lengthy. Once the application loads, unless there is a lot of required database access, the performance will be excellent. For e-commerce purposes, the load time for the applet is likely to be unreasonable.

In addition, the actions that the Java applet can perform on the client are restricted by the built-in security mechanisms. If those features are circumvented, Java applets can be written to perform tasks on the client machine such as writing to the file system. In many organizations, client machines reside behind a firewall that prohibits the downloading of Java applets.

Another disadvantage is that an applet uses an HTML browser to start up the JVM. Browsers do not universally support the Swing class libraries, and these are the basis for the Oracle Data Aware Controls (DAC) that have strong ties to the business components (BC4J) layer. If you do not use DAC, you are not taking advantage of one of the greatest strengths of JDeveloper and probably are writing too much code. AWT classes are fully supported but require much more coding to connect to the data layer. The impact on applet technology is that you must ensure that your users can access and install the Swing class plug-in offered by Sun. This requires a one-time step to download and install the plug-in.

Although one of the advantages of the applet architecture is its use of the web application server to store and serve the HTML file and applet class files, this is also one of its disadvantages. The application server requires additional configuration and maintenance, which, in turn, requires extra human resources and skills. While this requirement is manageable, it is one that must be taken into account.

Creating Applets in JDeveloper

The development process for applets in JDeveloper is very similar to the development process of Java applications. The components, layout managers, and events are the same so the same steps described above for Java applications apply equally to applets. You can specify a new applet in the Project Wizard's project type page. The Project Wizard will then launch the Applet Wizard after the project file is created. Alternatively, you can run the Business Components Data Form Wizard using **File→New** and specify an applet instead of a frame as the container for the data form. (This is not an option if you select Applet or Business Components Data Form in the Project Wizards type page.)

The considerations for selecting the data form or applet wizards are the same as mentioned above for the Java application.

When you run the Applet Wizard (from the Project Wizard or **File→New** for an applet), you will be able to specify that you want to insert the plug-in code into the HTML startup file. This code will check the client machine to see if the Sun Java runtime for Java 2 (that supports Swing and Oracle's DAC classes) is installed in the browser. If it is not, the user will be sent to the Sun web site to download the plug in.

One difference between working with Java applications and applets in JDeveloper is that to test an applet, you have to run an HTML file. If you run the HTML file that the wizards create for an applet, JDeveloper will emulate the browser call to the HTML file without using your browser. You can deploy the applet in a Zip or Java Archive (JAR) file and test this deployment without JDeveloper. In this situation, you run the HTML file by opening it in your browser. In the JDeveloper emulation, applet security is bypassed so the applet can freely write to the client's file system. Outside of JDeveloper, the standard applet security features are implemented. Therefore, you need to create a signature file if the applet needs to write to the client's file system.

JavaServer Pages

JSPs are different in many ways from Java applications and applets. They do not require a JVM on the client and they output HTML for display in a browser. JSP technology is an extension of servlet technology so it is useful to examine what a servlet is before discussing JSPs. A s*ervlet* is a program stored and run on the web application server that accepts requests from a client browser through an HTTP data stream (posted data or URL) and constructs an HTML page by querying the database and outputting the HTML tags mixed with data from the queries. The entire page is constructed dynamically by the program in a similar way to a Common Gateway Interface (CGI) program.

The advantage of servlets over CGI programs is that they only require a new thread, not an entirely new process like CGI programs. This is a significant resource saving for the application server. In addition, unlike CGI output, servlets are cached, which provides performance benefits such as allowing the database connections to stay open. Servlets are coded entirely in Java and are therefore portable and do not need a CGI language such as Perl.

JSPs are variations on servlet technology that mix HTML language and Java language in the same source file. They have both a dynamic and static element to them usually represented by the Java and HTML code, respectively. This allows the developer to easily code the parts of the application that do not change. For example, the JSP code would include the `<HTML>` tag at the beginning and `</HTML>` tag at the end of the page. It would also include all of the static links and boilerplate graphics and text.

A servlet has to generate these tags (using a `println()` statement) each time the program is run, whereas a JSP program represents the static tag exactly as it will be output. In reality, the HTML tags in JSPs are converted to Java `println()` calls so that they can generate a pure servlets when they are compiled and run, but the cleanliness of the JSP code justifies the choice of the JSP style over the servlet style.

Here is an example of the default JSP code that JDeveloper creates when you select JSP from the Object Gallery (**File→New**):

```
<%@ page contentType="text/html;charset=WINDOWS-1252"%>
<HTML>
<HEAD>
<META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=WINDOWS-1252">
<META NAME="GENERATOR" CONTENT="Oracle JDeveloper">
<TITLE>
Hello World
</TITLE>
</HEAD>
<BODY>
<H2>The following output is from JSP code:</H2><P><% out.println("Hello World"); %></P>
</BODY>
</HTML>
```

This sample mixes standard HTML tags ("< >") and JSP tags ("<% %>"). The file extension .jsp indicates to the web server that the page requested is a JSP file. The web server passes the interpretation of the page to a *JSP container* program that runs in a JVM on the server. The JSP container processes the JSP-specific tags, some of which may create additional HTML-formatted output. The container then mixes the static HTML tags with the HTML output generated by the JSP tags and sends the entire page back to the browser. Figure 7 shows the main elements of the JSP runtime architecture.

The first time a JSP page is accessed, the server process creates a Java servlet file and compiles that file into bytecode in a .class file. For subsequent accesses, the .class file is cached on the server so that this compilation is not required unless the code is changed. The JSP container runs the .class file in its JVM session. The Java and class files are generated dynamically from the JSP source code file. The BC4J layer sits on the application server and communicates with the database as in the other models. Figure 8 shows the various JSP code elements and the browser interaction.

When to Use JSPs

JSPs are indicated when your requirement is a simple, lightweight client with no firewall limitations. You would use them anywhere you would use standard CGI-generated or static HTML pages. If you can restrict your application to the limitations of the HTML and JavaScript languages, JSPs are a logical choice. Since this solution is more efficient on the server side, you can support a large number of users such as for an e-commerce application.

Advantages of JSPs

The main advantage of the JSP method is that the output is standard HTML and is therefore compact and universally readable in any browser. HTML requires little from the client except a compatible browser. There is no JVM running on the client so that there is no requirement for a set of Java runtime files or Java application files on the local PC.

The presentation look-and-feel of a page is embedded in HTML tags and cascading style sheets. Since the HTML tags are directly embedded in the JSP source file, you can split the development work. Creating the template look-and-feel for a page may be accomplished by a web graphics designer, while the dynamic JSP-specific sections would be developed by a Java programmer. Merging the work is just a matter of embedding one in the other. JDeveloper provides the tools to create and test the JSP code. The web designer would work in another tool such as Microsoft FrontPage or Macromedia Dreamweaver.

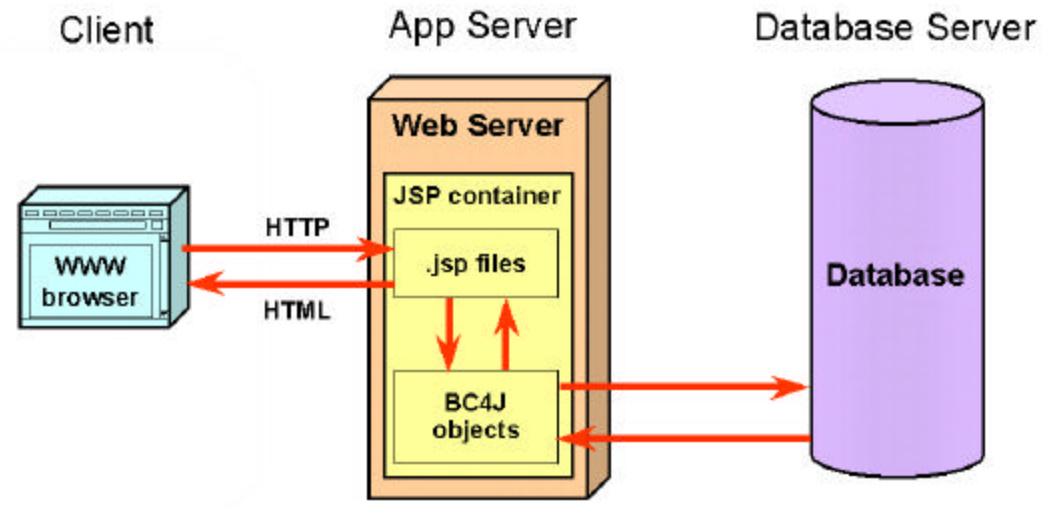


Figure 7. JSP architecture

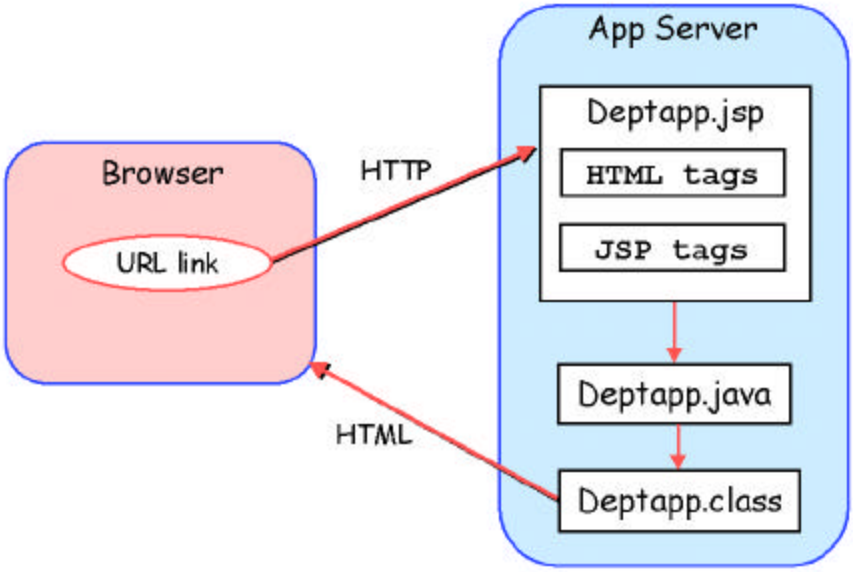


Figure 8. JSP code elements and browser interaction

Disadvantages of JSPs

The main advantage of JSPs—that they output lightweight HTML—is also the main disadvantage. You do not use the AWT, Swing, or Oracle DAC controls to construct the user interface. The HTML language has fewer features than the Swing and AWT controls for creating a user interface. In addition, simple functions such as scrolling down in a list of records, deleting a record or changing the way information is sorted requires a reload of the page. You can embed JavaScript in the HTML page to enhance functionality, but this solution requires that the JavaScript that you use be supported by the ability of your users' browsers to interpret it correctly.

Developing robust JSP applications requires that the developer (or at least someone on the development team) is skilled in Java, HTML, and JavaScript. For developers accustomed to using a single language for all coding, this will feel like a step

backwards. Debugging is more difficult because the code is running on the server in the JSP container. JDeveloper offers remote debugging features that assist in troubleshooting JSP applications.

The HTML limitation may not be important if you keep it in mind when deciding which technology to use for a certain application. There are many HTML applications on the World Wide Web that show reasonable complexity and suitability to their business functions.

Another disadvantage of JSPs are in the added complexity of the JSP tags and the architecture. There is added complexity in setting up the web server to support the servlet API and the JSP container. This extra complexity is not insurmountable but can be a daunting task for a shop that has not been involved with web technology. On the development side, JDeveloper includes a wizard (the JSP Element Wizard) that helps the developer embed JSP tags into the code. This wizard helps ease the learning curve that accompanies learning the JSP tags.

The Data Web Bean Solution

A JDeveloper feature called *data web beans* eases the limitation that JSPs cannot use AWT and Swing components. A data web bean (a data-aware, web-enabled JavaBean) is a type of web bean—a JavaBean that outputs HTML. The data web bean accesses the BC4J layer and outputs HTML containing data values from the database. For example, there is a data web bean that creates a single record display of a table in the BC4J application module.

Including JSP calls to a web bean runs the bean on the server when the JSP is accessed. This, in turn, retrieves the necessary data and outputs the HTML tags required to display the data in a browser.

The web bean contains properties that are set in the code you write to tailor its behavior. There are also methods in each web bean that you can call to alter the bean's behavior or features. You can write web beans that employ JavaScript for additional functionality.

The following is a sample tag structure that embeds the JSNavigationBar bean into a JSP file. This bean outputs an HTML version of the NavigationBar class found in the Swing components.

```
<jsp:useBean   class="oracle.jbo.html.databeans.JSNavigatorBar"  id="deptNavBar"
scope="request" >
<%
   deptNavBar.setShowNavigationButtons(true);
   deptNavBar.setReleaseApplicationResources(false);
   deptNavBar.initialize(pageContext,
      "testBeanPRJ_deptempbuscomp_DeptempbuscompModule.DeptView");
   deptNavBar.render();
%>
</jsp:useBean>
```

Figure 9 shows this navigator bar in action with a single record browser bean.

Creating Java Applications in JDeveloper

Developing JSP applications in JDeveloper is inherently different from developing Java applications and applets. Java applications and applets have a graphical design element to them. You drag and drop objects from the component palette into the Structure Pane or UI Designer. You can view and manipulate the properties of the components using the UI Designer and Inspector.

When you use JDeveloper to work with a JSP application, you do not use a visual editor; in JDeveloper, you will not use the Structure Pane, UI Designer, Inspector, or component palette. Most of the work that you do is in the Source Code Editor. The major visual part of the application results from HTML content that will most likely be developed using another tool such as Dreamweaver or FrontPage. These tools can be used in combination with JDeveloper so that you can work on both visual design and JSP construction. A web designer can work separately on the complex layouts using these tools and you can later merge her or his work with the JSP work that you have done in JDeveloper.

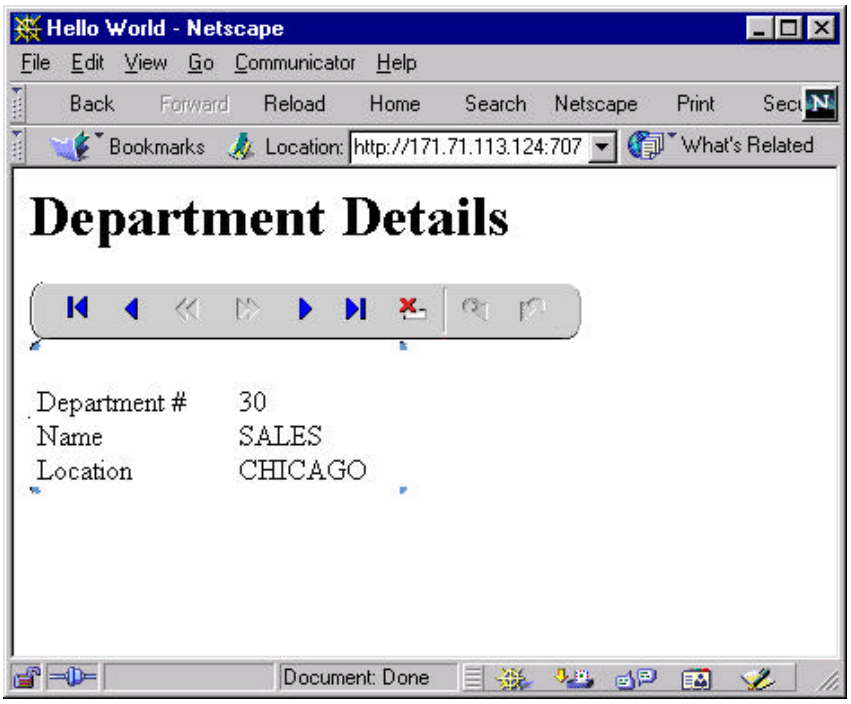


Figure 9. JSP navigator bar and record browser

More Skills Required

Work in JSP applications requires a broader range of knowledge than is required for work in Java applications or applets, for which you only need to know the Java language (in addition to database languages). In addition to knowledge of JSP tags, a typical JSP application requires familiarity with the following:

- HTML
- Java
- JavaScript
- Cascading style sheets

All of these are very approachable languages, but may require a bit of study and a good reference book before you become fully fluent and comfortable with development work in them.

JSP Development Steps

The normal working process in JDeveloper consists of the following steps:

1. **Create a text file using File→New** and selecting JSP in the Web Objects tab. This creates a default file with some sample JSP elements.
2. **Rename the file** so that when you save and compile it the file will have a meaningful name.
3. **Develop an HTML template or look and feel.** The JSP file that you create can incorporate an HTML template that contains the standard look-and-feel of your site. The HTML template might include a standard background graphic, a company logo in a standard position, a navigation bar containing links to the major areas of your site, a footer graphic, and links to contact information. The HTML layout work will be done in an appropriate tool such as FrontPage.
4. **Add JSP elements** that comprise the code using the JSP Element Wizard (**Wizards→JSP Element Wizard**). A *JSP element* is a piece of code that can be made up of HTML-like tags or other symbols. It is interpreted by the JSP container running on the server.

The main types of code that you will add to the JSP file are the following:

- **HTML tags** These are standard tags used to create browser pages. They are written into the JSP file just as they would be written into a normal HTML file.
- **Standard JSP tags** JSP elements use the same kind of tag syntax as HTML tags, but contain special characters (such as % signs) to distinguish them from HTML tags. These are standard tags created by Sun and documented on their web site.
- **JDeveloper JSP elements** JDeveloper provides several categories of elements that assist in application development, such as JSP custom data tags, web beans, and data web beans. There is also a supplied set of themes, or cascading style sheets, that make the visual elements of a JSP application consistent.

5. **Test the file** using JDeveloper's built-in web server emulator *Web-to-go*. The pages will be displayed in your browser, and you will be able to check the design, interaction with the database, and user interface functionality. Make notes of the modifications needed to comply with the requirements.
6. **Modify the code and test it again.** This modify/test step continues throughout the development process.
7. **Deploy the files** to another server so that they may be available to users.

JDEVELOPER JSP ELEMENTS

JDeveloper provides three main categories of JSP elements:

- **BC4J Data Tag Library** (called Data Tag Library in the JSP Element Wizard) These tags provide complete access to database operations at a low level using the BC4J layer. Some tags can also display data in a rudimentary way.
- **Web beans (HTML web beans)** Web beans are JavaBeans that output HTML. The HTML web beans have no links to the database. You need to combine these beans with the Data Tag Library to present data in a JSP file.
- **Data web beans** These JavaBeans also output HTML and allow you to define data elements as attributes of the web bean. Therefore, you do not need to separate the data and user interface areas—they are both part of the data web bean.

These categories are represented by folders in the JSP Element Wizard. Each category implies a different level of coding that requires a different method as described next.

Methods for Creating JSP Applications

There are many ways to create JSP applications in JDeveloper. The following is a list of some of the most common development methods ranked from hardest and most time consuming to easiest and fastest.

- **Hand code using the Source Code Editor.** You would use the BC4J Data Tag Library to retrieve data for the HTML display.
- **Use web beans.** Web beans save you from having to manually code the HTML that displays a particular control object. To retrieve data from the database, you would have to combine the web bean code with the BC4J Data Tag Library tags.
- **Run the DataPage Wizard.** The DataPage Wizard (available through **File→New,** Web Objects tab) constructs pages one at a time based on the BC4J Data Tag Library tags. This is just a more automatic way of getting the same results as the previous method would provide.
- **Use data web beans.** These beans automatically include database access and display capabilities, so you can more quickly build a data-aware page. The amount of code that you have to write is much less because you do not have to worry about the data source.
- **Run the Business Components JSP Application Wizard.** This wizard creates an entire application quickly from a view object. The problem is that modifying the application is a bit of work because you have to understand how the generator constructs its code.

As you would expect, the quicker and easier methods at the end of this list are less flexible. Hand coding always yields the most flexible method. However, the quicker and easier methods generate code that is completely modifiable, so you can always get what you want from those methods. The best and fastest method for most beginners who still want some control is the data web beans method.

A key technique for learning how JSPs are constructed is to run the Business Components JSP Application Wizard and spend time examining its output to see where you might tap into the code and how you might code a JSP file on your own. Another key technique for learning is to examine the sample Online Orders application that is included with JDeveloper. It contains a working application that makes extensive use of the JDeveloper JSP tags and you can gather much information about JSP techniques from this sample.

> **Note**
>
> Another paper by this author titled "A JDeveloper Primer for Oracle Forms Developers" in the ODTUG proceedings discusses the differences and similarities between JDeveloper and Oracle Developer. After reviewing this discussion, you may choose to deploy an application using Oracle Developer rather than Java and JDeveloper.

Conclusion

There are many alternatives when developing and deploying applications that you create using JDeveloper. Java code is flexible enough to run in client/server as well as in web environments. This paper has explained the details, benefits, and drawbacks for three common application deployment styles—Java applications, applets, and JSPs. You will have to weigh various factors in your decision such as the following:

- Complex functionality that needs to be supported
- Scalability
- Performance
- Ease of installation and maintenance

The information in this paper should help you decide how to select one of these styles for a specific purpose, and, once you have made the decision, understand the architecture of the chosen technology and how to start the development work.

About the Author

Peter Koletzke is a Technical Director and Principal Instructor for the Enterprise e.Commerce practice at Quovera (formerly Millennia Vision), in San Jose, California. Peter is Executive Vice President and Web Content Director for the IOUG-A and columnist for the *ODTUG Technical Journal*. He is a frequent speaker at various Oracle users group conferences where he has won awards such as Pinnacle Publishing's Technical Achievement, ODTUG Editor's Choice, and the ECO/SEOUC Oracle Designer Award. He is the co-author, with Dr. Paul Dorsey (who contributed to this paper) of the Oracle Press (Osborne McGraw-Hill) books: *Oracle JDeveloper 3 Handbook* (which supplied material for this paper), *Oracle Developer Advanced Forms and Reports*,*Oracle Designer Handbook, 2nd Edition*, and *Oracle Designer/2000 Handbook*.
http://ourworld.compuserve.com/homepages/Peter_Koletzke

Quovera provides strategy, systems integration, and outsourced application management to Fortune 500, high-growth middle market and emerging market companies. The firm specializes in delivering intelligent solutions for complex enterprises, which improve productivity within the customer's business and optimize the customer's value chain, through integration of its customers and suppliers. The company also outsources the management of "best of breed" business applications on a recurring revenue basis. Quovera refers to its business model as "Intelligent – Application Integration and Management."
http://www.quovera.com