JDEVELOPER 10G VIEW LAYER ALTERNATIVES —JSP AND UIX

Peter Koletzke, Quovera

Java 2 Platform, Enterprise Edition (J2EE) offers many alternatives for coding a user interface. The most popular J2EE interface for web applications lately has been JavaServer Pages (JSP) files, which is coded in a mixture of HTML and Java tags. Oracle offers an alternative, called ADF UIX that defines the page using XML instead of HTML. UIX provides some benefits to JSP pages but it has many similarities. Development of both of these alternatives is well supported in Oracle JDeveloper 10g.

This paper explains these two view layer technologies and describes how to develop them using Oracle JDeveloper 10g—Oracle's Java Integrated Development Environment (IDE). It reviews the principles of the MVC design pattern and how work in JDeveloper follows that design pattern. This paper also discusses JSP and UIX architectures, code libraries, coding styles, benefits and drawbacks, key features, and intended uses. In addition, it provides conclusions about which style is best for specific situations. Additionally, the paper will describe how to develop an application in both styles using JDeveloper.

MVC

J2EE consists of the Java language, specifications, guidelines, and best practices. One of the key components of J2EE is J2EE design patterns that describe low-level design solutions to common design problems. One of the most pervasive and popular of all J2EE patterns is *Model-View-Controller* (MVC), which is actually inherited from the Smalltalk language. JDeveloper's ADF is built from the principles inherent in MVC and it is important to understand these principles when working in JDeveloper.

MVC defines a rigorous separation between these three components of application code (shown in Figure 1):

- **Model** This layer represents the data and values portion of the application.
- **View** This layer represents the screen and user interface components.
- **Controller** This layer handles the user interface events that occur as the user interacts with the interface (view), controls page flow, and communicates with the Model layer.

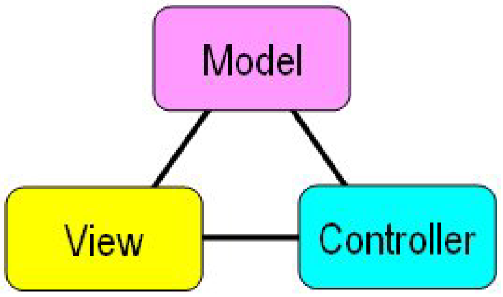


Figure 1. MVC layers

The controller layer of MVC is often implemented in a centralized servlet that is responsible, among other things, for defining and implementing *page flow*—which page will be called from a specific action on another page. Before the incorporation of MVC principles into web technology, page flow was handled in a decentralized way; that is, each page would contain a hard-coded link to another page or would pass a parameter value to the next page, which processed the value to determine what would be displayed. The Java community refers to this decentralized method as *Model 1* and the centralized controller method as *Model 2*. Although those terms were only used in draft versions of the JSP specification, they are still in common use.

The separation between layers allows the switching of one code layer without affecting the other layers. For example, if your application were built using the MVC pattern, you could switch the user interface from a Java application to a mobile cell phone interface and still use the underlying controller and model layers. An extension of this principle is that a single model and controller layer can support more than one view at the same time, for example, in an application where some users accessed your application by cell phone and some by a desktop computer. This kind of flexibility is the key benefit of the MVC pattern.

The separation of layers also allows different developers to work on different parts of the application code. In addition, compartmentalized design work for data structures, interface screens, and processes affects code in different MVC layers—model, view, and controller, respectively.

Therefore, in theory, MVC offers many benefits. In practice, this separation is rarely achieved completely but it is still viable as a goal and a design principle.

MVC IN JDEVELOPER 10G

Oracle JDeveloper 10g offers assistance in development of any J2EE code. It provides a development architecture called *Application Development Framework* (ADF) that is built around the MVC design pattern. The components of ADF, shown in Figure 2, are divided into four layers that are patterned from MVC. ADF splits the Model layer into two: the Model layer and the Business Services layer. This split provides even more reusability when a technology changes or when you need to plug in another technology.

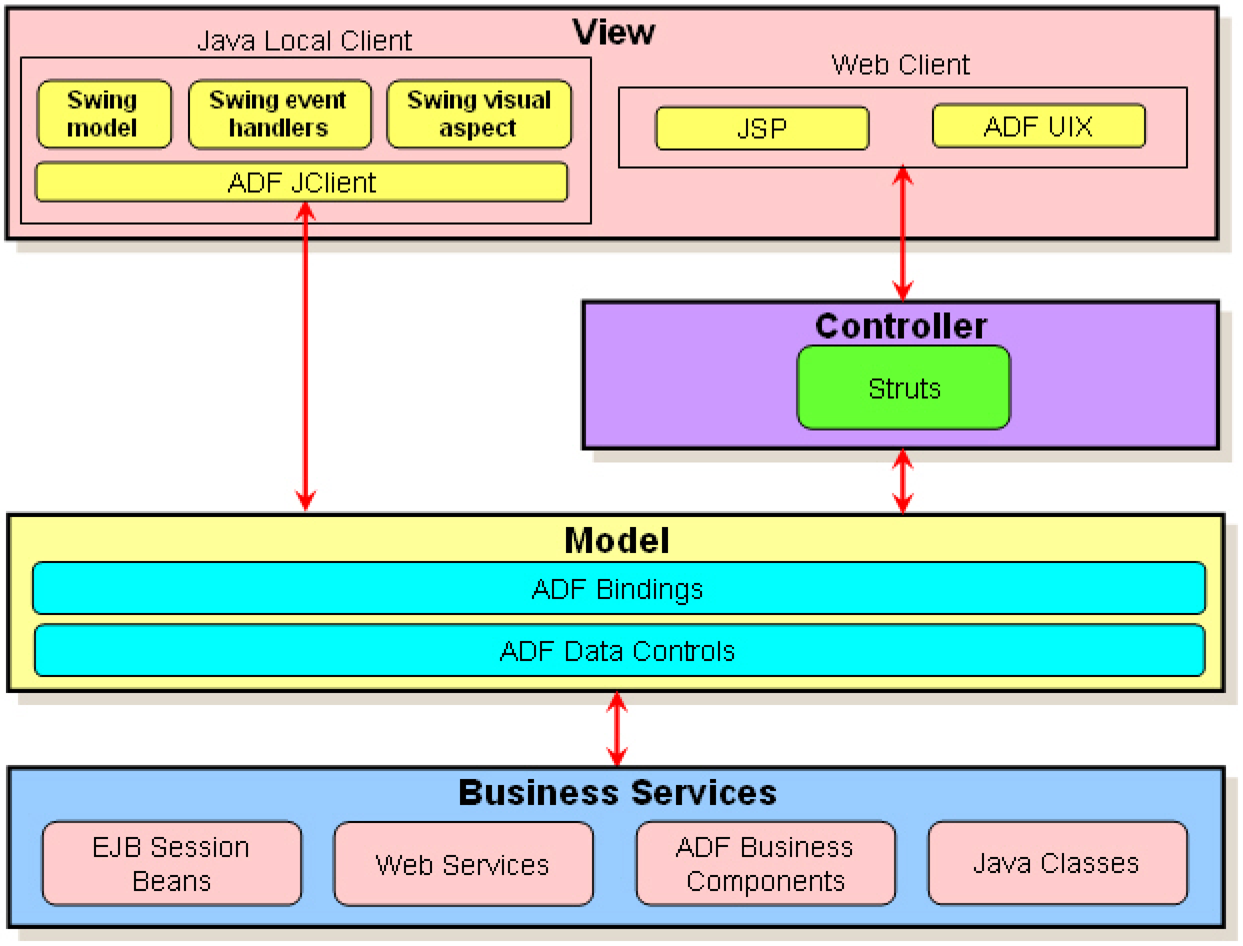


Figure 2. Oracle ADF Technologies

Several of the technologies in this diagram are worthy of a brief mention because JSP and UIX code requires code in the Controller and Business Services layers.

STRUTS

Struts is an open source framework developed and maintained by the Apache Software Foundation (jakarta.apache.org/struts). It is a set of tag libraries, code libraries, and strategies for creating logic that defines page flow and data communications from the View layer to the Business Services/Model layers. Although it is not an Oracle framework, it is the preferred and recommended method for developing controller code using JDeveloper 10g (at the moment).

ADF BUSINESS COMPONENTS

The ADF Business Services layer contains a framework, *ADF Business Components* that assists in the task of connecting your application code to the database. ADF Business Components (ADF BC) allows you to define Java classes that correspond to database tables and views so that your Java code can access the database in a more natural (for Java), object-oriented way.

More detailed explanations of the ADF architecture are available on Oracle Technology Network (otn.oracle.com). Since this paper concentrates on the View layer technologies, a brief explanation of the ADF View layer is in order, however.

ADF VIEW LAYER

The ADF View layer consists of two main areas: *Java Local Client* (Java code running in a runtime session on the client machine) and *Web Client* (Java code running in a runtime session on a remote server with the user interface rendered in a light-weight client such as a web browser). The technologies on which this white paper focuses fall into the Web Client category: JSP and UIX.

JAVASERVER PAGES (JSP)

JSP technology is an evolution of Java servlet technology and it is necessary to understand the parent technology.

JAVA SERVLET TECHNOLOGY

A *Java servlet* is a Java class file that is stored and run on the application server and that extends the functionality of a server by providing extra services or application-specific logic. An *HTTP servlet* is a specific type of Java servlet (and a subclass of the Servlet class) that accepts requests from a client browser through an HTTP data stream (posted data or URL) and forms a response (usually in HTML format) that is sent to the browser. The servlet executes within a *container*—a service that can run code in a JVM. The *web container* (on the Web Tier) runs servlets and JSP code, and the *EJB container* (on the Business Tier) runs EJBs. Each container processes the target code in a specific way. More information about the J2EE tiers may be found in the sidebar "What are J2EE Tiers?".

An HTTP servlet (often just called "servlet") can construct an HTML page by querying the database and outputting the HTML tags mixed with data from the query. The entire page is constructed dynamically by the program.

Before Java web technologies, a common way to present dynamic HTML content was to use a common gateway interface (CGI) program written in the Perl language. The advantage of servlets over CGI programs is that servlets only require a new thread, not an entirely new process like CGI programs. In addition, the memory and state of multiple threads are managed by the servlet runtime. This is a significant resource saver for the application server. In addition, unlike CGI output, servlets are cached, which provides performance benefits such as managing the state of database connections. Servlets are coded entirely in Java and are therefore portable and offer a bytecode "compiled" runtime version.

What are J2EE Tiers?

The J2EE standards outline an architecture comprised of logical *tiers*— areas where code and processes reside. J2EE architectural tiers consist of the *Client Tier* (where user interface code runs), the *Web Tier*, (a remote server where user interface code is constructed), *Business Tier* (where validation and business logic code is run), and the *EIS Tier* (where the database is located).

JSP TECHNOLOGY

JSP technology is a variation on servlet technology that mixes HTML and Java in the same source file. JSP pages have both dynamic and static elements, usually represented by the Java and HTML code, respectively. This allows developers 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 the `</html>` tag at the end of the page. It would also include static links, boilerplate graphics, and boilerplate text.

A servlet generates 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 JSP page is compiled into a servlet .java file. The clarity of the JSP code provides developers with an advantage over the servlet style. In addition, because pure HTML code appears in the JSP page, you can use a visual HTML editor to work on the layout of the JSP page.

The following is an example of the simple JSP file:

```
<%@ 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 (delimited by "`< >`") for static content such as headings with Java servlet tags (delimited by "`<% %>`") for dynamic content created by running code in Java classes.

About Tag Libraries

In addition to standard tags such as `page` shown in the example, custom tags can be created. Each tag points to a Java class file that is collected into a Java archive (JAR) file. *Tag definition files* are XML files that contain pointers from the tag name to the Java class. The collection of tags used for a specific purpose is called a *tag library*. JDeveloper offers many fully-featured tag libraries that facilitate your JSP work.

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* (web container) program that runs in a servlet process on the server. The JSP container runs the JSP file through a *JSP translator* program to convert file into a pure-Java, servlet file (.java) by adding `out.println()` statements for the HTML tags and adding print statements and other Java code for the JSP-specific servlet tags. The servlet is then compiled (to a .class file) and run in the container's JVM, and the servlet output is sent to the browser as HTML.

Figure 3 shows the main elements of the JSP runtime architecture in the context of the J2EE tiers. For JSP technology, the Web Tier runs the JSP container process and waits for a request from the browser.

When the request for a JSP page appears, the web server determines that the request is for the JSP container because the file name has a .jsp extension. The JSP container process (JVM) runs the application file and accesses the application code as well as the controller code. These layers access code on the Business Tier (ADF BC in this example), which accesses the database. After the code is run and the HTML page is constructed, the server sends the page back to the browser, and the browser displays it as it would any other HTML page.

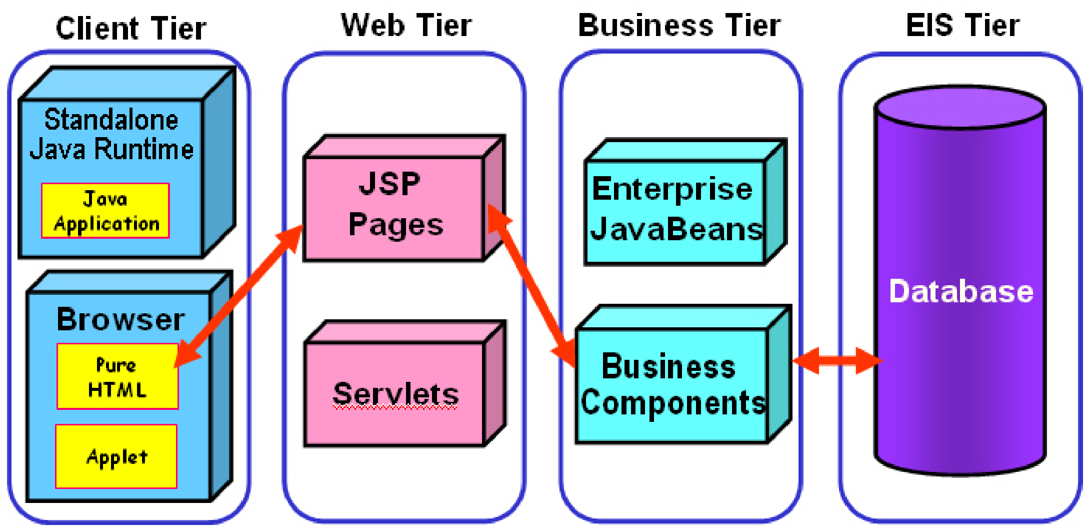
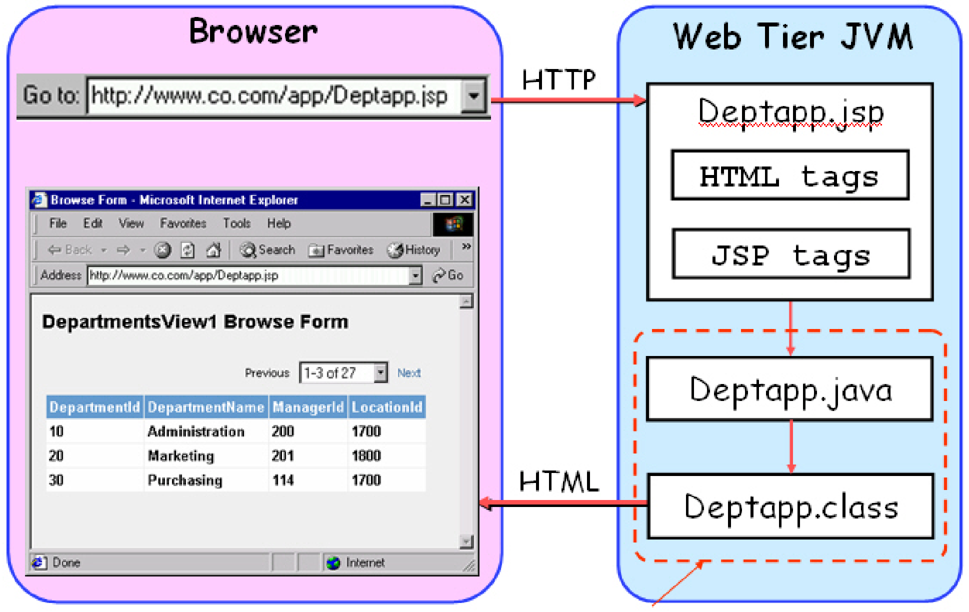


Figure 3. JSP Runtime Architecture

Note

Although Figure 3 shows JSP files and servlet files separately, when you use JSP technology, you are also using servlet technology because the JSP file is translated into a servlet. However, when speaking about the technology you are using, you would mention JSP technology because you do not code the servlet directly.

The first time a JSP page is accessed, the server process translates it into a Java servlet file (as mentioned before) 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 Business Tier layer sits on the application server and communicates with the database as in the other models. Figure 4 shows the various JSP code elements and the interaction with the browser. The dotted-line box represents a one-time compilation.


Browser
Go to: http://www.co.com/app/Deptapp.jsp
HTTP
HTML
Web Tier JVM
Deptapp.jsp
HTML tags
JSP tags
Deptapp.java
Deptapp.class
Browse Form - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://www.co.com/app/Deptapp.jsp Go
Previous 1-3 of 27 Next
Done
Internet

DepartmentsView1 Browse Form

DepartmentId	DepartmentName	ManagerId	LocationId
10	Administration	200	1700
20	Marketing	201	1800
30	Purchasing	114	1700

Figure 4. JSP calling sequence

Therefore, JSP applications are different in two basic ways from Java Local Client applications: They do not require a JVM on the client and they output HTML (or other types of content) that is displayed in a browser.

Strengths of JSP Pages

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 only requires a compatible web browser on the client machine. There is no JVM running on the client, so there is no requirement for a set of Java runtime files or Java application files on the local machine.

The presentation look-and-feel of a page is embedded in HTML tags and *cascading style sheets* (an HTML facility for changing the appearance and formatting of common tags in a standardized way). Since the HTML tags are directly embedded in the JSP source file, you can split the development work. A web graphics designer can use HTML to create the look-and-feel for a page, while the dynamic JSP-specific sections would be developed by a Java programmer. Merging the work is just a matter of embedding one into the other. JDeveloper provides the tools to create and test the JSP code. It also provides a visual editor, toolbars, Component Palette pages, and Property Inspector support for the visual layout of the JSP file.

Limitations of JSP Pages

The main advantage of any web client technology including JSP pages—that they output lightweight HTML—is also a limitation because you do not use the feature-rich Swing (or AWT) controls to construct the user interface. HTML is a text presentation language and is not a programming language as such. Therefore, it has fewer features than the Swing controls for creating a highly interactive user interface. In addition, simple functions such as scrolling through a list of records, deleting a record, or changing the way information is sorted require a refresh of the page. This limitation is lessened somewhat by embedding JavaScript in the HTML page to enhance functionality if the users' browsers support JavaScript. In addition, the HTML limitation may not be important if you keep it in mind when deciding which technology to use for a certain application. Many HTML applications on the World Wide Web show reasonable complexity and suitability for complex business functions.

Required Skills

Developing robust JSP applications requires standard web development skills including an expert level of HTML knowledge as well as a working knowledge of Java and JavaScript. In addition, it is useful to have a handle on CSS (Cascading Style Sheet) language that allows you to more easily create a common look and feel for your application. For developers accustomed to using a single language for all coding, this will feel like a step backward because the primary code environment is 3GL. However, development tools such as JDeveloper can assist with the generation of this code and provide you with a visual and declarative interface most of the time.

Note

The Worldwide Web Consortium maintains standards and documentation for HTML and CSS. You can start learning more information about those languages at w3c.org. Start learning about JavaScript from tech.irt.org (the devedge.netscape.com website that acted as the starting point for JavaScript learning has been phased out).

Web Technology Complexity

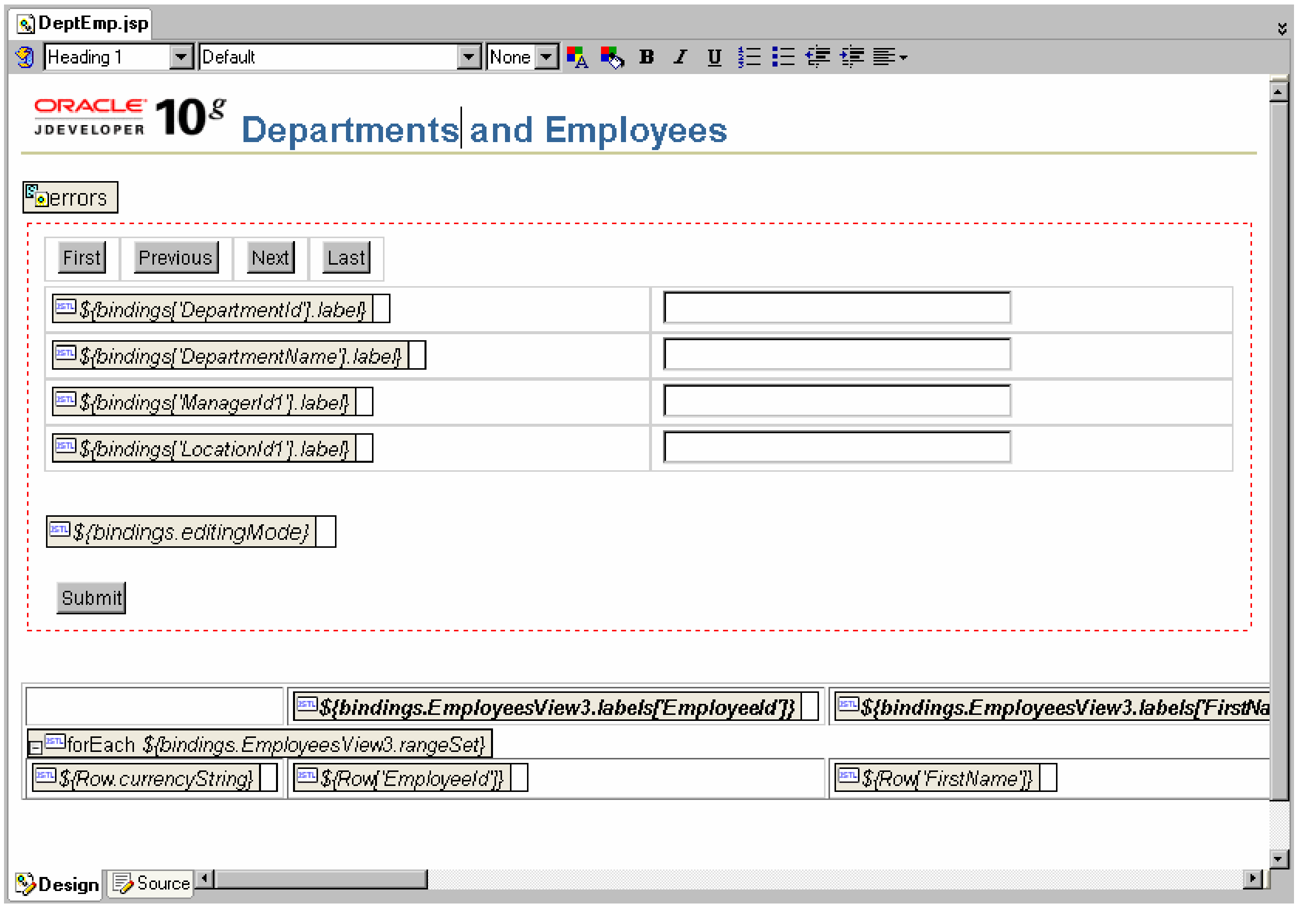
Another disadvantage of any web solution including JSP pages is in the added complexity of the tags and the architecture. There is also added complexity in setting up the web server to support the servlet API and the JSP container. This extra complexity is not insurmountable and is made easier by standard tag libraries, Java tools such as JDeveloper, and J2EE architecture standards.

Developing JSP Applications in JDeveloper 10g

JDeveloper 10g offers tools to assist in design, development, and deployment of code in all three MVC layers. For the View layer, JDeveloper provides visual layout tools, wizards, component palettes, code libraries, and editor features that help you create JSP pages. In addition, when you develop JSP applications in JDeveloper, you use many of the same tools as when you develop Java client applications (Java applications and applets) in JDeveloper including the Component Palette, Data Control Palette, visual editor, Property Inspector, Structure window, and the Code Editor. It is useful to examine the key features of each tool used for JSP page development and the steps for creating a JSP application.

JSP/HTML Visual Editor

The JSP/HTML Visual Editor (shown in Figure 5) allows you to perform HTML editing in an environment that emulates the appearance of the JSP page at run time. Double clicking a JSP file in the navigator or Page Flow Diagram (described later) will display this editor. In addition, if you are working with the Code Editor for the file, clicking the editor's Design tab will display the visual editor.



Heading 1 | Default | None

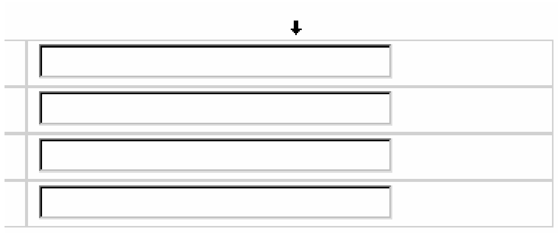
```
${bindings['DepartmentId'].label}
${bindings['DepartmentName'].label}
${bindings['ManagerId1'].label}
${bindings['LocationId1'].label}
${bindings.editingMode}
${bindings.EmployeesView3.labels['EmployeeId']}
${bindings.EmployeesView3.labels['FirstNa
forEach ${bindings.EmployeesView3.rangeSet}
${Row.currencyString}
${Row['EmployeeId']}
${Row['FirstName']}
```

Figure 5. JSP/HTML Visual Editor

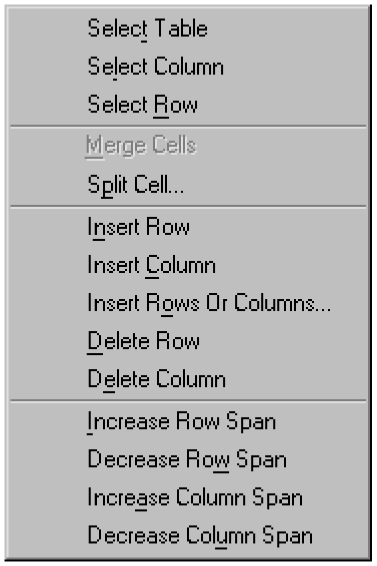
This visual editor displays fonts, colors, graphics, and all HTML tags as they will be displayed in the browser. In addition, tag library tags, such as the Struts errors tag shown in Figure 5, will be represented by gray boxes labeled with the tag name or value and an icon indicating the tag library where the tag is defined (such as JSTL). As with all tools in JDeveloper, the visual editor is synchronized with the Structure window, Property Inspector, and Code Editor so that changes in any of the tools will be reflected immediately in the other tools. In fact, if you select an element in the visual editor, the code that defines the element will be highlighted in the Structure window and Code Editor (described later).

The JSP/HTML Visual Editor shares the following features in common with separate, commercially available HTML editors:

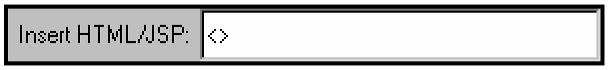
- **A toolbar with formatting tools** for applying HTML styles, fonts, font sizes, colors, text attributes (bold, italic, and underline), paragraph styles (numbered list, bulleted list, indent, and outdent), and alignment (left, center, and right).
- **Drag-and-drop operations** on its elements so that you can easily position elements.
- **Features for HTML table editing** such as the following:
    - **Table, column, and row selection** by holding the mouse next to the column or row as shown next and clicking the mouse button.



- **Right-click menu options** as shown next that appear when the cursor is in an HTML table in the Table submenu.



- **Smart delete actions** so that when you select a row or column in an HTML table and press DELETE, the HTML representation of the row or column will be deleted along with its contents.
- **Manual addition of any tag** by typing in the tag using the Insert HTML/JSP dialog, which appears as follows after you select Insert HTML/JSP from the right-click menu:



- **Tag editing** using the editor dialog available for a tag by selecting Edit Tag from the right-click menu.

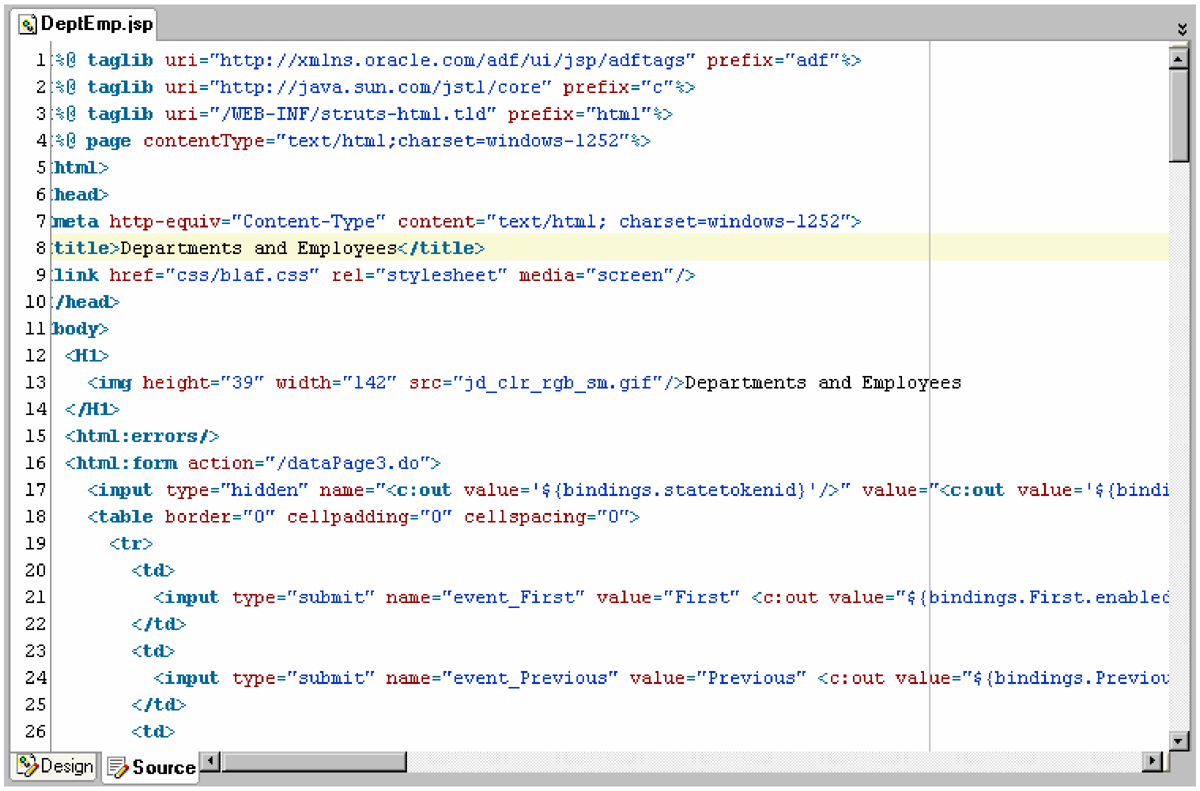
- **HTML form actions** in the right-click menu for selecting, deleting, or changing the form method (`Get` or `Post`).
- **The ability to accept images** dropped in from a file system utility (such as Windows Explorer), from the navigator, or from a web browser.

Tip:

To add image (and other types of) files to a project, you can drag the file from another program such as Windows Explorer or Internet Explorer and drop it onto the project node in the navigator. This operation does not move the file, but if you subsequently drag the image onto a page, a dialog will appear asking if you want to copy the file to the project directory.

Code Editor

Although most of the work you perform will be in the visual editor or Page Flow Diagram, you can and will probably need to edit text in the Code Editor (available by clicking the Source tab in the editor window after a JSP file is opened) as shown here:



```
<%@ taglib uri="http://xmlns.oracle.com/adf/ui/jsp/adftags" prefix="adf"%>
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c"%>
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html"%>
<%@ page contentType="text/html;charset=windows-1252"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
<title>Departments and Employees</title>
<link href="css/blaf.css" rel="stylesheet" media="screen"/>
</head>
<body>
 <H1>
   <img height="39" width="142" src="jd_clr_rgb_sm.gif"/>Departments and Employees
 </H1>
 <html:errors/>
 <html:form action="/dataPage3.do">
   <input type="hidden" name="<c:out value='${bindings.statetokenid}'/>" value="<c:out value='${bindi
   <table border="0" cellpadding="0" cellspacing="0">
     <tr>
       <td>
         <input type="submit" name="event_First" value="First" <c:out value="${bindings.First.enabled
       </td>
       <td>
         <input type="submit" name="event_Previous" value="Previous" <c:out value="${bindings.Previou
       </td>
       <td>
```

The Code Editor is a configurable text editor with extensive support for 3GL coding. In addition to standard text editing, the Code Editor offers features for JSP editing such as the following:

- **Code highlighting** that shows tags, attributes, comments, and user-defined values in different colors.
- **Syntax checking**, which is performed as you are editing. Errors are displayed in the Structure window before compiling.
- **End Tag Completion** fills in tag endings (such as "`</h2>`") for HTML and JSP tags after you type "`</`".
- **Tag Insight** for tags presents a list of valid attributes for the tag element and for Java code inside *scriptlets* (Java fragments inside JSP tags).
- **Dragging and dropping of tags** from the Component Palette and Data Control Palette into any location in the JSP file.
- **Attribute editing** in the Property Inspector, which is available after you place the cursor anywhere inside a tag in the Code Editor.

Many of these features are configurable in the Preferences dialog (**Tools | Preferences**).

Tip

Context-sensitive help is available in the Code Editor for the custom JSP tags supplied with JDeveloper. For example, if you place the cursor inside a Struts tag such as "`<html:errors/>`" and press F1, the help topic for the errors tag will be displayed.

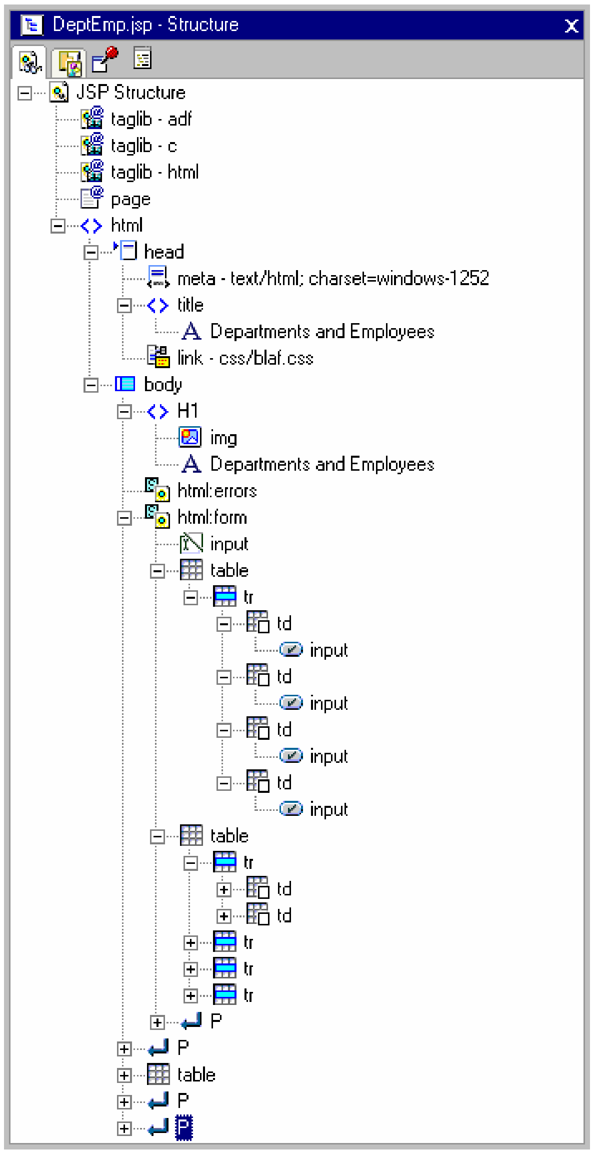
Structure Window

The Structure window works with the Code Editor to display organizational information about the file. The Structure window contains two tabs: the UI Model tab, which shows the contents of the UIModel.xml file and the data model elements used in the file; and the JSP Structure tab (shown in Figure 6), which displays the hierarchy of HTML and other tag elements in the file.

The Structure window also shows a representation of the tag attributes and their values. This gives you an accurate picture of how the JSP page is constructed. As with other types of files, you can click a node in the Structure window to navigate to its code in the Code Editor or its visual representation in the visual editor. The Property Inspector shows and allows you to edit the properties of the element selected in the Structure window.

Tip

The JSP Structure tab of the Structure window allows you to drag and drop nodes to change the code.



```
DeptEmp.jsp - Structure
JSP Structure
    taglib - adf
    taglib - c
    taglib - html
    page
    html
        head
            meta - text/html; charset=windows-1252
            title
                Departments and Employees
            link - css/blaf.css
        body
            H1
                img
                Departments and Employees
            html:errors
            html:form
                input
                table
                    tr
                        td
                            input
                        td
                            input
                        td
                            input
                        td
                            input
                table
                    tr
                        td
                        td
                    tr
                    tr
                    tr
                P
            P
            table
            P
            P
```

Figure 6. JSP Structure tab for a JSP file

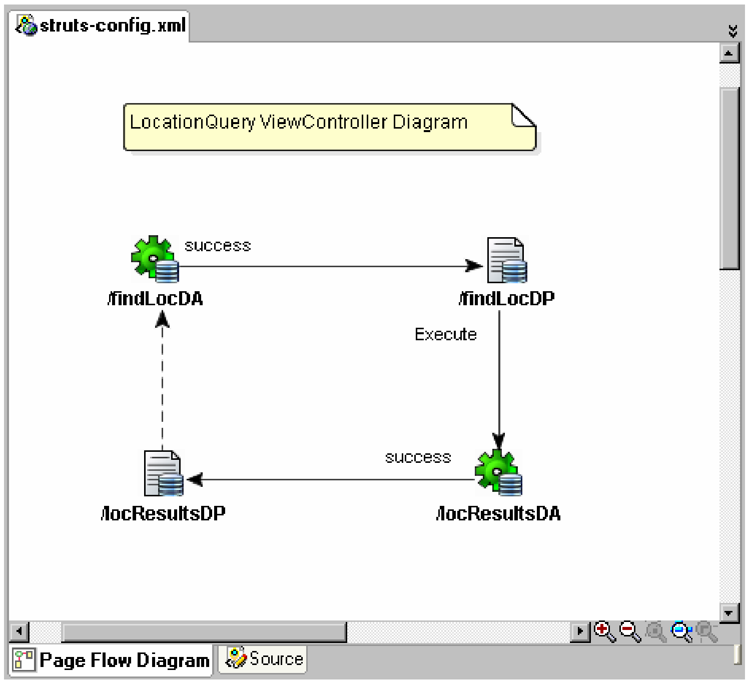
The Structure window also offers the following features:

- **Toolbar buttons for Freeze and New Value**. The Freeze button allows you to prevent the Structure window from highlighting a selected element and New Value allows you to open a new Structure window. Freezing and showing new views are useful if you want to compare the contents of one window (frozen) with another (unfrozen) window that shows a selected element.
- **Tag editing** using the Edit Tag, Cut, Copy, Paste, and Delete right-click menu options.
- **Table and form actions** as in the visual editor.

- **Dragging and dropping of tags** within the Structure window so that you can easily restructure the JSP file.
- **Addition of tags** from the Component Palette or Data Control Palette by dragging and dropping to, above, or below a node in the Structure window.

Page Flow Diagram

The Page Flow Diagram, shown next, is a key tool for JSP applications that use the Struts controller (the preferred style in JDeveloper 10*g*) because it allows you to quickly build and modify the struts-config.xml file in a graphical way.



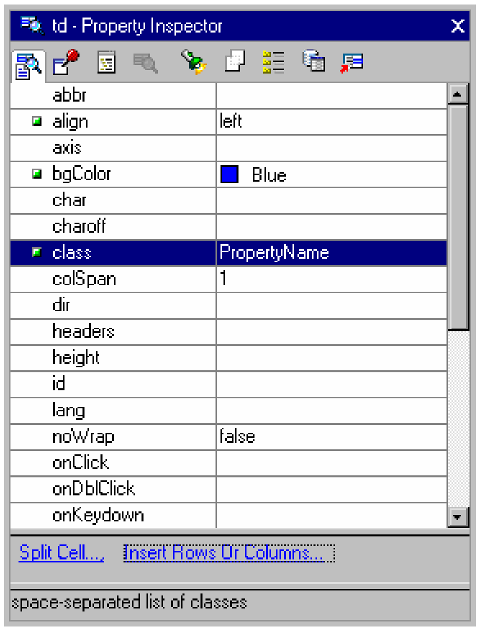
The Page Flow Diagram, unlike most windows, is not available from the View menu, but you can display it by double clicking a struts-config.xml file in the navigator, by selecting Edit Struts-Config from the right-click menu on that file, or by selecting Open Struts Page Flow Diagram from the right-click menu on the project node in the navigator.

The Thumbnail window, a separate window that automatically appears when you open a page flow diagram, shows a miniature picture of the diagram. It allows you to quickly navigate a large diagram by dragging a view window box around the Thumbnail window.

Property Inspector

The Property Inspector, shown next, allows you to edit properties or attributes of tags in the file.

You can access the properties of an object by selecting it in the Structure window, JSP/HTML Visual Editor, or Code Editor. Grouping elements together allows you to apply property values to all elements at the same time. Toolbar buttons allow you to find properties and display the properties with category headings. For a group of selected elements, the Union button displays the intersection (only common properties) or the union (all properties in all elements). As with
the Structure window, you can use buttons in the toolbar to freeze the property list or to create another window containing properties.


td - Property Inspector

abbr	
align	left
axis	
bgColor	🟦 Blue
char	
charoff	
class	PropertyName
colSpan	1
dir	
headers	
height	
id	
lang	
noWrap	false
onClick	
onDblClick	
onKeydown	

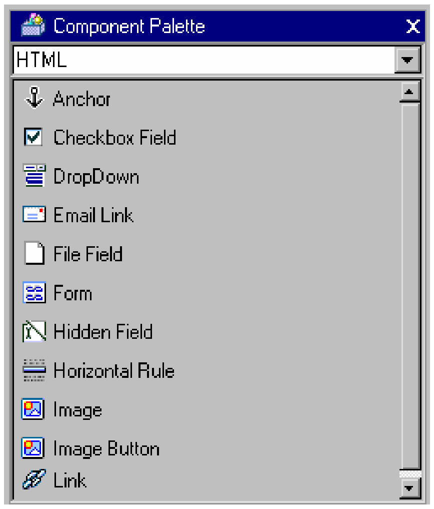
Split Cell... Insert Rows Or Columns...

space-separated list of classes

Component Palette

The Component Palette (shown in Figure 7) allows you to drop tags into the visual editor, Code Editor, or Structure window. After selecting a page from the Component Palette pulldown, you can select an element and place it in the proper place in one of those three tools. If the tag requires property settings, the appropriate property dialog will appear so that you can type in values before the tag is created.

When you create a JSP file, the appropriate Component Palette pages will be available. If needed, you can add tags to the Component Palette. However, the tag libraries available to a JSP should offer enough functionality for most requirements.


Component Palette

HTML

- Anchor
- Checkbox Field
- DropDown
- Email Link
- File Field
- Form
- Hidden Field
- Horizontal Rule
- Image
- Image Button
- Link

Figure 7. Component Palette for HTML tags

Generally, you decide on one type of tag library or framework for a particular project because each implies a specific method of operation and often a specific look-and-feel. All of the data-aware frameworks and libraries are capable of accessing data through ADF business services, and JDeveloper makes binding to data sources relatively simple.

One of the main strengths of a tag is that it can incorporate a large amount of functionality. Since a custom tag is based on a Java class file, the base class file can encapsulate complex functionality into a single line of code in the JSP page. You can adapt or create tags to fit the needs of your project. Custom tags fall into the category of JSP action tags. Therefore, they are just single-line calls to Java class files, rather than tags that include low-level Java code (as are
the standard tags included with the JSP engine such as the scriptlet, expression, and declaration).

Note

All elements in the Component Palette offer tooltip hints that you can use to get an idea of the purpose of the tag. Although some hints are briefer than others, they should all help you determine which tag is best for a specific purpose.

DATA CONTROL PALETTE

The Data Control Palette, shown in Figure 8, provides components that are bound to business services objects. The mechanics of inserting components from this tool are similar to the drag-and-drop operation you use with the Component Palette. The difference with the Data Control Palette is that you select the data control model element first from the list at the top of the window. Then you select a component from the *Drag and Drop As* pulldown at the bottom of the window. Finally, you drag the model element into the Code Editor, JSP/HTML Visual Editor, or Structure window.

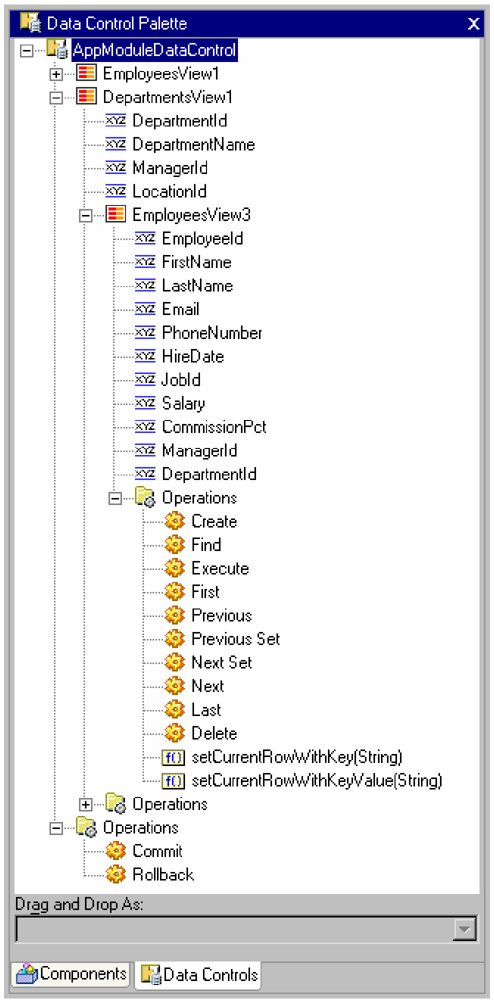

```
Data Control Palette
AppModuleDataControl
  EmployeesView1
  DepartmentsView1
    DepartmentId
    DepartmentName
    ManagerId
    LocationId
    EmployeesView3
      EmployeeId
      FirstName
      LastName
      Email
      PhoneNumber
      HireDate
      JobId
      Salary
      CommissionPct
      ManagerId
      DepartmentId
      Operations
        Create
        Find
        Execute
        First
        Previous
        Previous Set
        Next Set
        Next
        Last
        Delete
        setCurrentRowWithKey(String)
        setCurrentRowWithKeyValue(String)
    Operations
  Operations
    Commit
    Rollback
Drag and Drop As:
Components   Data Controls
```

Figure 8. The Data Control Palette

The result of dragging components from both palettes is similar—a tag or set of tags appear in the code. However, the Data Control Palette binds the component automatically to the data model, whereas you need to manually bind the components you drop in from the Component Palette.

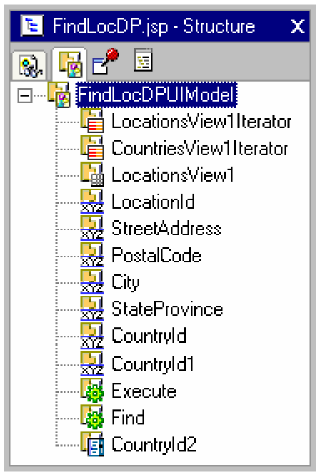
THE DATA MODEL

The data model in the Data Control Palette is based upon the business services (in this example, ADF BC application modules) in the Model project. The hierarchy of the data model will be displayed in the palette as shown with EmployeesView3 under DepartmentsView1 in Figure 8. This data model also contains an EmployeesView1, which is not linked to DepartmentsView1. When selecting the data model node, it is important to keep in mind which view object instance (linked or independent) is required for a specific page.

The Data Control Palette offers tags at three main levels: data model level (for business services), data collection level (for example, ADF view object definitions such as DepartmentsView1 in Figure 8), and attribute level (such DepartmentId in Figure 8). In addition, the Data Control Palette offers *operations*—actions that work on the collection level (for example, view object operations such as `Create`, `Find`, `Delete`, `Execute`, `First`, `Last`, `Next`, `Previous`) or on the model level (application module operations such as `Commit` and `Rollback`).

DATA BINDING EDITORS

When you drag a data control into a JSP file, the tools set a data binding for each tag that requires data. This data binding can be viewed and modified in the UI Model tab of the Structure window as shown here:



FindLocDP.jsp - Structure

- FindLocDPUIModel
  - LocationsView1Iterator
  - CountriesView1Iterator
  - LocationsView1
  - LocationId
  - StreetAddress
  - PostalCode
  - City
  - StateProvince
  - CountryId
  - CountryId1
  - Execute
  - Find
  - CountryId2

In most cases, the binding created by the tools will work without modification. However, some controls require more complex settings and you will need to use the binding editors available in the Structure window. Various editors exist for various binding types (for example attributes, tables, iterators, actions, and lists). Each binding is given a name and this name is referenced in the tag that displays the data.

DATA CONTROLS FOR STRUTS JSP PAGES

Although the *Drag and Drop As* list changes based upon what element is selected and what type of file is being built, it is useful to examine the controls available for different levels of the data model.

MODEL LEVEL

The model level for an ADF BC project is named with the application module name and the suffix "DataControl." Use these controls when you need to act on the database transaction. For example, a button control is available when selecting the `Commit` or `Rollback` operations under this node. This button signals a server operation that will affect the database transaction. An HTML form is required to process any HTML button. If you drag a button into a location in the JSP file that is not surrounded by an HTML form, you will see a dialog that asks if you also want to create an HTML form. Clicking Yes in this dialog will add a form element around the button you are dropping.

Note

`Commit` and `Rollback` can also be dropped as a method onto a component in a page flow diagram.

DATA COLLECTION LEVEL

The collection level for an ADF BC application module corresponds to the view object instance in the data model of an ADF BC application module. These controls are used when you need a representation of a row from the data source. Some of the data controls available when a collection level (such as DepartmentsView1 in Figure 8) is selected are: Read-Only Table (an HTML table with column headings, data cells, and a current row indicator), Navigation Buttons (a table with submit buttons for first, next, previous, and last row operations), and Input Form (an HTML form containing a table with labels in one column, fields in another column, and a submit button). Controls available when selecting one of these operations on the collection level are the same button and button with form as on the model level.

Attribute Level

The final level for data controls is the attribute level of the collection. This level represents a single data value within the collection (such as LocationId under DepartmentsView1 in Figure 8). Some of the data controls available to the attribute level of the model are: Value (a JSTL out tag that displays data), Label (a JSTL out tag that displays text), Input Field (an input tag), and List of Values (a Struts select tag).

Note

The help system topic "About UI Components in Oracle ADF Web Pages" shows how these controls will be displayed in a JSP page and further describes the ADF binding. This topic is available by searching for "About UI Components" in the help system's Full Text Search tab.

Steps for Creating a Struts JSP Application

JSP applications that use the Struts controller and ADF BC business services use the following general steps to create the application:

1. Create a business services project containing the data collections required for the pages.
2. Create a Struts page flow diagram containing the actions, pages, and forwards.
3. Drop the required data controls from the Data Control Palette onto each page.
4. Modify the layout of the elements by dragging and dropping them in the Structure window or visual editor.
5. Adjust properties as required using the Property Inspector.
6. Modify or define the data binding for elements if needed.
7. Adjust the visual display by adding cascading style sheets and graphics, applying styles, and changing colors and fonts.
8. Test the application by selecting Run from the right-click menu on the starting file in the Page Flow Diagram. This starts up the Embedded OC4J Server, which is a "real world" container process for JSPs. This server is the same one used to run JSP pages within the Oracle Application Server. Therefore, running it in JDeveloper replicates the production environment.

ADF UIX

Initially, the UIX framework, formerly called User Interface XML (uiXML), was used by developers of the Oracle E-Business Suite (Oracle Applications) for front-end, self-service applications and by customers of the E-Business Suite to develop extensions to those front ends. In addition, Oracle uses UIX as a front end to products such as Oracle iLearning, Enterprise Manager (EM), and Oracle9*i* Internet File System (iFS). Now, JDeveloper offers the ability to easily create UIX applications inside or outside of the Oracle application products.

Note

More information about ADF UIX is available in the Online Demos, Tutorials, Code Samples, and How-To's sections from the JDeveloper home page on OTN (otn.oracle.com/products/jdev). Additional overview and reference material is contained in the JDeveloper help system. In the Search tab, look for "oracle adf uix developer's guide" and double click "Oracle ADF UIX Developer's Guide," which explains the framework in detail. In the Table of Contents tab, look under the Reference book node for "UIX Reference" topic, which describes the UIX tags.

UIX Features

ADF UIX offers many features that distinguish it from traditional J2EE technologies.

Platform Independence

UIX code files use a .uix extension and are built with XML syntax using the ADF UIX tags. The use of XML means that the code is interpreted at run time (in contrast with JSP code, which is translated and compiled before it is run). It also allows the language to support deployment to different platforms. If the display device does not support HTML, UIX can be instructed to render the display in a form that the device can accept. This allows UIX files to be displayed in multiple output devices, such as desktop computer browsers or mobile devices. Naturally, the UIX design must take the device type into account, so a particular UIX file will be designed for a particular output device. For example, the page must be much simpler if the output will be on a PDA because the screen is smaller than on a desktop monitor.

Note

JDeveloper offers a UIX JSP tag library that allows you to include many UIX features in a normal JSP file. This tag library is not recommended for new applications because of planned obsolescence and because it requires an extra layer in the runtime library process.

Integrated into ADF

UIX is completely integrated into the view layer of the ADF framework. This means that UIX code can be developed in JDeveloper in the same way as other types of code that ADF supports. It also means that UIX code can share the controller and business services layers with other types of code. UIX can bind to any ADF data source.

Internationalization Support

UIX supports internationalization (abbreviated as *i18n*)—the application is adaptable to any language. Image files are created dynamically (as described in a later section) so that they can be built for a particular language at run time. In addition, styles are defined in XSS (XML Style Sheet Language). At runtime, UIX generates a locale-specific CSS (cascading style sheet) based on the definitions. This style sheet can be defined specifically for a language, if needed.

UIX and J2EE

Although UIX is an Oracle-specific technology, it is J2EE-compliant, proven, and mature. It shares with JSP technology the ability to render an HTML page from Web Tier code. In addition, UIX shares design principles with JavaServer Faces (JSF) technology, which was recently ratified by the Java Community Process as Java Specification Request 127 (jcp.org/en/jsr/detail?id=127). Future releases of UIX will implement JSF principles and act as extensions to JSF functionality. For more information about JSF and ADF, refer to the white paper "Roadmap for the ADF UIX technology and JavaServer Faces" currently available in the Technical Papers section of JDeveloper's home page on OTN (otn.oracle.com/products/jdev).

Standardization

UIX uses a page design model (described later) that provides consistency within the application. The template use in UIX enforces standards use as well. In addition, the concept of *look-and-feel*, a standard set of fonts and colors, is integrated into the page display. All of these features help make application pages appear in a consistent way.

Rich Component Set

UIX contains a large variety of user interface components with rich functionality such as the shuttle control that you can use to move items between lists or reorder items in a list. Components are divided into the following categories:

- **Simple components** are low-level, user input controls such as fields, buttons, and links. Many of these controls offer features more powerful than those offered by normal HTML controls. For example, the `messageDateField` tag displays a field and a button; when the user clicks the button, a calendar appears in a separate window and allows the user to select a date, which is then passed back to the field.
- **Layout components** act as containers for other components. These components correspond roughly to Java client containers such as JPanel and their layout managers such as BorderLayout and FlowLayout. Each UIX layout component that contains other components defines a layout style such as `borderLayout` or `flowLayout`. In fact, the UIX layout components often have an exact counterpart in the Swing layout managers.

- **Composite components** are higher-level collections of simple and layout components. These collections are prebuilt to perform a special operation, such as the `tree` element that displays a hierarchical data in a navigator format.

As with any tag in a tag library, the rich functionality of these components makes developing basic modules for each new application easier. Developers do not need to think in terms of HTML coding. Rather, they think in terms of larger units. In addition, templates and look-and-feel attributes aid in creating a pleasing and user-friendly page design. Although you can code JSP tags to implement virtually any look-and-feel, the default UIX templates and tags are richer-looking out of the box.

UIX Dynamic Images

UIX creates image files on the fly for objects such as buttons and tabs. The text property value of the tag (such as the tag for a button) is derived from the object at run time and the graphic is generated accordingly. This dynamic image generation further facilitates internationalization because the text value for an object can be customized to the locale or language in which the page is displayed. Also, developers need not worry about creating or properly locating graphics files for the design objects such as tabs.

Easy Look-and-Feel Swapping

UIX ships using the Oracle Browser Look-and-Feel (BLAF) design layout, which was developed for the Oracle E-Business Suite. You can switch the look-and-feel to another Oracle look-and-feel or to one of your own design. A single UIX can be displayed in any look-and-feel available by setting a single property in the UIX configuration file.

Extensibility

You can extend the capabilities of UIX using these two facilities:

- **Templates (UIT)**, which are reusable files with a .uit extension that combine standard components to create a new component. A template will be displayed as a tag in the UIX file and, when run, it will be included in the UIX page. After creating a template, you will see it automatically appear in the Component Palette. JDeveloper offers no visual editor for templates but does offer two template wizards and support in the Code Editor. Templates are a powerful feature for standardizing the content or look-and-feel of an application.
- **Extensions**, which are custom renderers (display controls) that fulfill complex requirements. You would develop these custom tags if templates were not sufficient or if no existing component is usable. Extensions require low-level coding and this should not be necessary often.

Partial Page Rendering

UIX offers the ability to perform *partial page rendering* (PPR), a refresh of part of the page. No custom JavaScript is required, although JavaScript is used to perform the contents update. The mechanism starts with a partial page event that results in only part of the page contents being sent back to the browser. PPR uses inline frames (*iframes*) to communicate between the browser and the web application. The UIX controls that support PPR are hGrid, hideShow, hideShowHeader, lovInput, processing, subTabLayout, and table.

In addition to these distinctive features, UIX uses a special tag structure that is useful to explore before starting to learn about developing UIX pages.

UIX Page Design Structure

Unlike JSP tags, which are often closely coupled with the HTML tags shown in the browser, a component in a UIX file usually translates into more than one HTML tag; the UIX tag places the HTML tags on the page in the order in which they appear in the code according to a specific page design structure. This structure is organized around a hierarchy of *user interface nodes*, nodes arranged in a hierarchical tree represented in XML by nested components.

User Interface Nodes

User interface nodes are logical structures that identify a particular part of the page or components within those areas. Figure 9 shows a design structure (from the Structure window) of a search page.

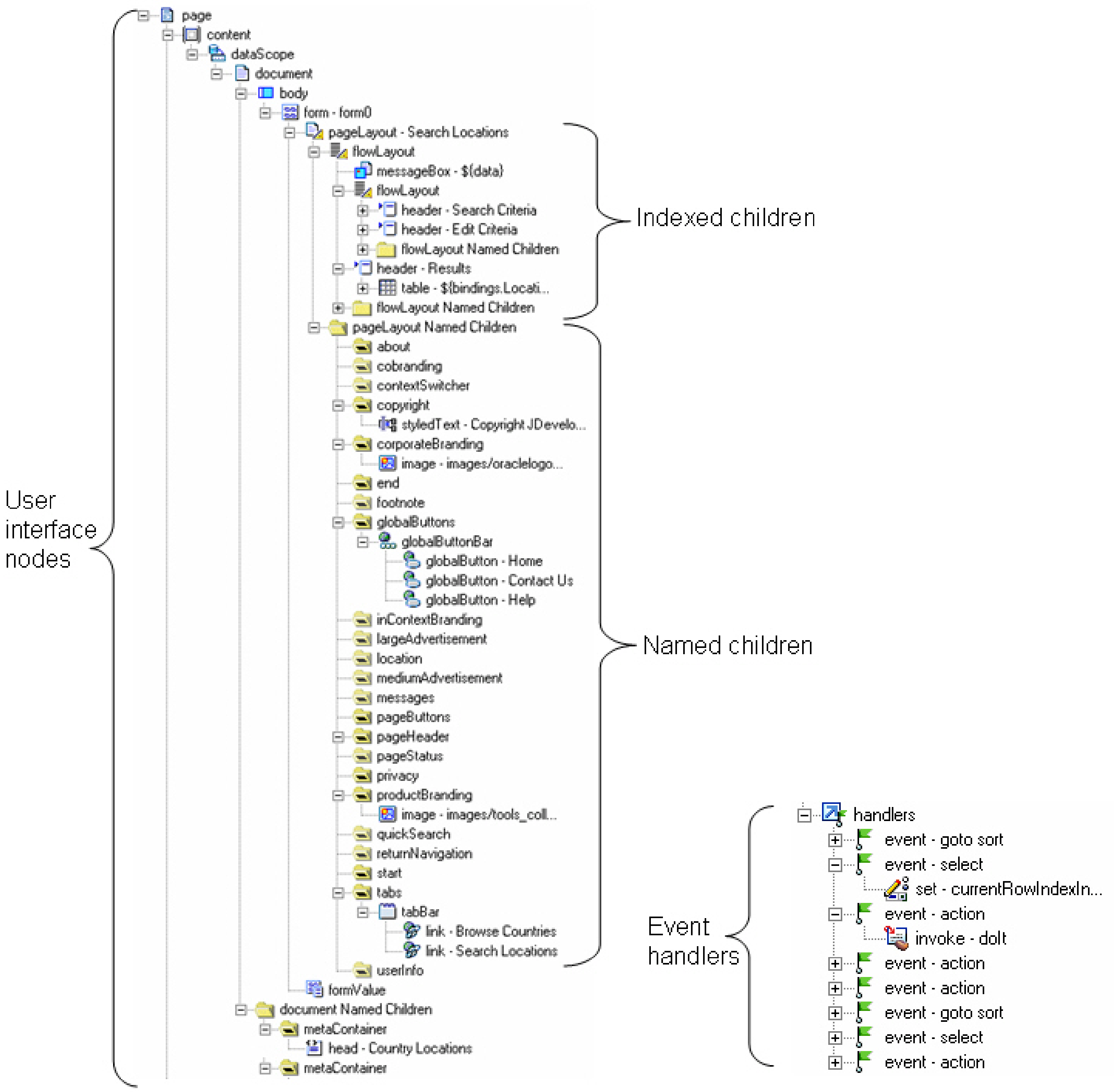


Figure 9. Sample UIX design structure

In this example, the page node is the parent user interface node for all other nodes defined for the page. The content and handlers nodes act as children of the page node and parents to their nested elements. Following the hierarchy down through the content node, you will see the form node representing the HTML form; under the pageLayout node, you will find a flowLayout node containing all components for the page.

The pageLayout node contains a number of preset *named children* slots (placeholders for specific components) including copyright, corporate branding, global buttons, pageHeader, product branding, and tabs as shown later in this section. It also includes named child nodes not being used in this file (those which have no "+" symbol next to them).

Therefore, you can place components in the named children nodes that will correspond to a specific location on the page. You can also stack components separately (as *indexed children*) outside of the named child nodes. The "pageLayout -Search Locations" node in Figure 9 shows examples of indexed children.

EVENT HANDLERS

In addition to page elements, the right-hand hierarchy shown in Figure 9 lists the events defined for the page. (This hierarchy appears under the page design hierarchy in the Structure window.) *Event handlers* define what happens for a specific user interaction. After defining the *event* property of an object, you create the event in the Design Structure tab and link the event to a specific controller action. In a project that uses Struts as the controller, this action could be a data action or a data page.

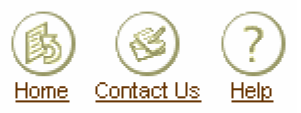
UIX COMPONENT EXAMPLE

Named child nodes have recommended contents, but there is no requirement to place a particular component inside a particular node. For example, the named child, globalButtons, usually contains a `globalButtonBar` tag that, in turn, contains `globalButton` tags.

The component tags render the HTML elements. For example, when you display a single UIX tag, `globalButtonBar`, in a browser, the tag translates into an HTML table containing the links and images defined inside the named child tag borders (from `<globalButtons>` to `</globalButtons>`). The code for such a structure in UIX would look like the following snippet from a UIX page:

```
<globalButtons>
  <globalButtonBar>
    <contents>
      <globalButton source="images/www_home.gif"
                    text="Home" destination="#"/>
      <globalButton source="images/www_contact.gif"
                    text="Contact Us" destination="#"/>
      <globalButton source="images/www_help.gif"
                    text="Help" destination="#"/>
    </contents>
  </globalButtonBar>
</globalButtons>
```

This code is rendered by the *UIX Servlet* (the runtime process that processes UIX code) in the following way:



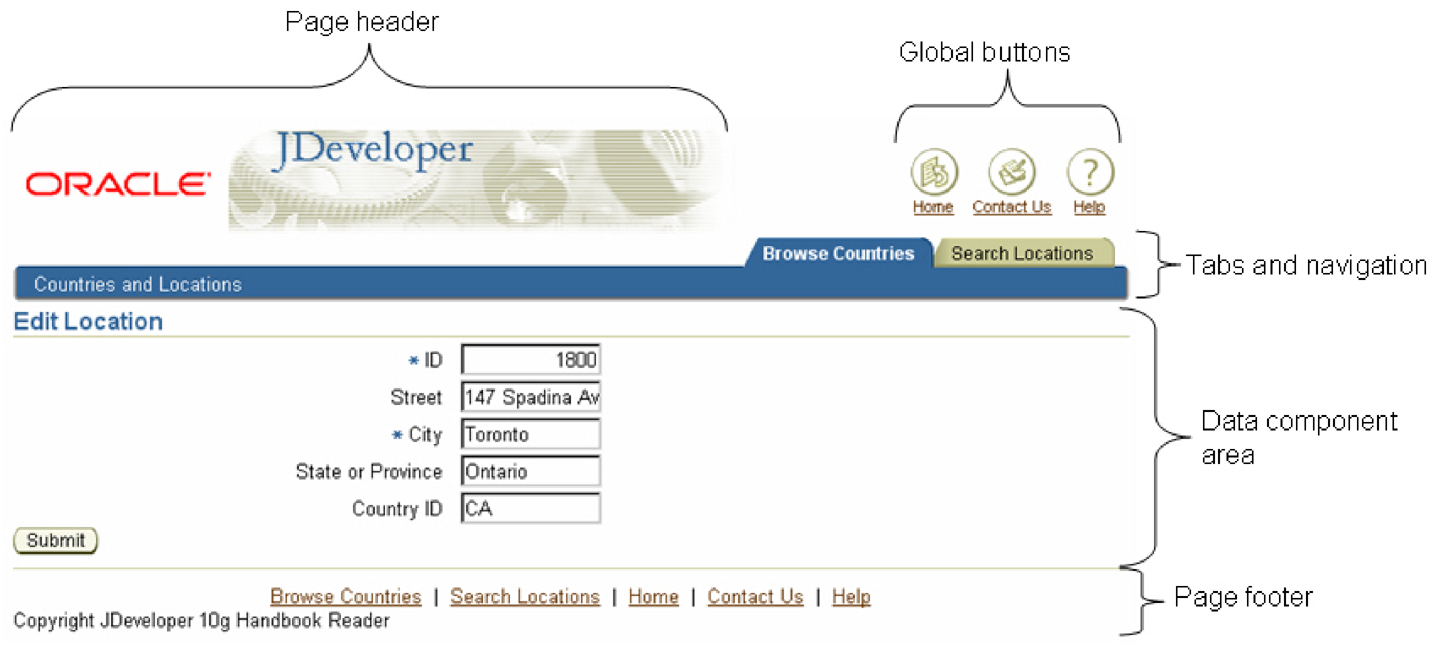
The translation of code tag to HTML display is the same as with a high-level JSP tag that can display an HTML table with two rows and three columns including graphics and text labels. UIX, however, also places this object in the position assigned to the globalButtons named child node (in this example, the top-right corner of the page). In addition, the same tag can be used to display the same component in a different output device such as a PDA.

NAMED CHILDREN EXAMPLE

Figure 10 shows the UIX Visual Editor with an edit page. This figure identifies the following page design areas, and Figure 9 shows the hierarchy nodes (for the search page) that correspond to some of these areas:

- **Page header** This area contains the *corporate branding* (a graphic that identifies the company or a top-level entity) and the *product branding* (a graphic that identifies the application or low-level entity). The corporate branding and product branding can also be stacked.
- **Global buttons** This area holds buttons that appear on all pages. The buttons link to other pages or websites.
- **Tabs and navigation** This area shows the tab control that allows users to quickly navigate to another page. In addition, the page can contain an additional navigation bar control (usually on the left side of the page) that contains links to more pages.

- **Data component area** This area is reserved for *data components*, controls that interact with data, for example, input forms or browse tables. You drop components from the Data Control Palette into this area to place data on the page.
- **Page footer** This area shows the *copyright* (a message at the bottom of the screen), the *privacy message* (the statement to the user about information sharing), and a *privacy statement* (the privacy text). In addition, the links from the tab area and global buttons are repeated in text in this area.



* ID	1800
Street	147 Spadina Av
* City	Toronto
State or Province	Ontario
Country ID	CA

Copyright JDeveloper 10g Handbook Reader

Figure 10. UIX Visual Editor with layout areas

Skills Required

As in JSP development, the developer needs a working knowledge of the capabilities of HTML and JavaScript. However, with UIX, the code consists of XML, not HTML so an additional requirement for a developer is knowledge of how the UIX tags will translate into HTML tags (as in the global button example shown before). Knowledge of JavaScript is less critical because most JavaScript is already built into the UIX controls. Coding in Java is not really required by the user interface, but if modifications are needed to the basic UIX controls, an expert knowledge of Java and JavaScript is required.

Developing UIX Applications in JDeveloper 10g

JSP and ADF UIX technologies both use code deployed in the Web Tier and a user interface displayed in a web browser. The tools that are used for JSP file creation and modification are used in a similar way with both technologies. Therefore, additional discussion about the basic tools and operations for using those tools is unnecessary.

The JDeveloper tools assist in the creation of UIX code and for much of the time, you will not need to type XML code in the Code Editor. The UIX Visual Editor, Property Inspector, Structure window, Data Control Palette, and Component Palette allow you to work in a declarative way. In addition, wizards help create files containing templates or standard named child container elements.

Tip

If you know JSP technology and would like to read about UIX in the context of JSP technology, on the Search tab of the help system, look for "about jsp pages and uix xml" and review the topic "About JSP Pages and UIX XML Similarities and Differences."

Wizards and File Creation Dialogs

The following list describes the New Gallery items for creating ADF UIX files. In some cases, the item displays a file dialog where you enter the name of the file and click OK. In other cases, a wizard appears and you fill out the wizard pages to define the file. Finishing the wizard creates the file.

- **Empty UIX XML Page** This item opens a dialog that creates a UIX file containing basic tags such as providers (data sources), body (content), and handlers (for events).
- **UIX XML Page Based on Existing UIX XML Template (UIT)** This item starts the "Create UIX XML Page Based on UIT Wizard," which creates a file by including a template for common objects such as images, copyright text, and header text. The wizard is relatively simple because all common page design areas have been defined in the template.
- **UIX XML Page with Header, Footer, and Navigation** This item starts the "Create UIX XML Page Wizard," which allows you to define the major page design areas (title, branding images, navigation elements, and footer element). It does not use a template.
- **UIX XML Page with pageLayout** This item starts a dialog where you specify a file name and click OK. The file that is created contains the same objects as the empty UIX XML page described before; in addition, a pageLayout tag and related tags are added so that you can add content without having to worry about many of the components.
- **UIX XML Template (UIT)** This opens a dialog that creates a basic template (UIT) file with no objects.
- **UIX XML Template (UIT) with Header, Footer, and Navigation** This dialog opens the "Create UIX XML Template (UIT) Wizard," which creates a UIT file containing tags for tab pages, footer text, global buttons, corporate branding, product branding, and page title. Once you define a template, you can use it as the basis for a UIX file with the New Gallery item "UIX XML Page Based on Existing UIX XML Template (UIT)."

UIX Visual Editor

The UIX Visual Editor allows you to view the components of a UIX page. As with the JSP/HTML Visual Editor, the UIX Visual Editor accepts drag-and-drop operations from the Component Palette and Data Control Palette. You do not type text directly into the visual editor; instead, you select an object and set its properties in the Property Inspector. The hands-on practice provides experience with the process of modifying content using property values.

Note

You can add, move, and delete objects in the UIX Visual Editor but you cannot perform inline editing (direct typing into the editor). Inline editing is planned for a future JDeveloper release.

UIX Preview

The UIX Preview is available by clicking the Preview tab. Although you cannot make changes to the file in this mode, you will see the page in the way that it will appear in the browser. The UIX Visual Editor shows the page in a similar way but it also shows named child areas that are empty.

Note

JSP files have no preview capability such as that available to UIX.

XML Editor

The XML Editor is available from the Source tab of the editor window when a UIX or UIT file is displayed. This editor contains the usual code-editing features including syntax highlighting, syntax error evaluation (errors appear in the Structure window), Tag Insight, and Tag Completion. Remember that tag completion for an XML file works differently from tag

completion for an HTML or JSP file.
The XML ending tag will fill in automatically after you type the "`>`" character for the starting tag. For example, as soon as you type "`<pageLayout>`" in the Code Editor, the "`</pageLayout>`" tag will pop in. For JSP and HTML files, you need to type the entire starting tag and the "`</`" characters of the ending tag (for example, "`<body></`") before the ending tag is completed automatically.

Tip

As in the JSP code editor, pressing F1 after placing the cursor in a tag will display the help system topic or Javadoc for that tag.

Structure Window

The Structure window for UIX files contains the following tabs:

- **XML Structure** This tab shows the hierarchy of XML tags (like the hierarchy of JSP tags in the JSP Structure tab for a JSP file).
- **UI Model** This tab displays the data binding to business services components and allows you to access the binding editors for each node. This works the same way as for a JSP file.
- **Design Structure** This tab (shown in an expanded view in Figure 9) displays the nested page layout elements in the UIX file. It is unique to UIX files.

The Design Structure tab allows you to see the possible named children that you can add to the page. It also allows you to use right-click menu options to add elements to user interface nodes. The right-click menu options "insert before," "insert inside," and "insert after" allow you to specify the relative position of the added element inside the hierarchy.

Page Flow Diagram

The Page Flow Diagram represents the struts-config.xml file and acts the same for a UIX application as it does for a JSP application. If you define a data page and double click its icon, the Select or Create Page dialog will appear. If the file name you fill in for this dialog uses a .uix extension, a UIX file will be created with standard UIX page tags. If the file name you fill in uses a .jsp extension, a JSP file will be created.

Property Inspector

The Property Inspector for UIX files looks and works the same way it does for JSP files although, in this release of JDeveloper, you cannot select a tag in the XML Editor and have the properties appear in the Property Inspector as you can in the code editor for JSP files.

Component Palette

The Component Palette for UIX files looks and works the same way as the Component Palette for JSP files. It contains pages such as All UIX Components, which lists all available UIX tags; Code Snippets, which allows you to create a Component Palette element for your own code; Form Components, which contains elements that will be placed in the HTML form for user input, such as messageDateField, messageLovChoice, and messageTextInput; and Include Components, which allows you to insert content from other UIX or non-UIX pages, for example, include, servletInclude, and urlInclude.

In addition to the prebuilt pages, other pages will appear for each template that you add to the project. The tab page will be created from the targetNamespace attribute of the templateDefinition tag and the element name will be formed from the localName attribute of the same property.

Note

More information about these groups of components is available in the JDeveloper help system "Building ADF UIX User Interfaces" topic. Search for those words in the Search tab and sort by Topic Title to find this topic.

Data Control Palette

The Data Control Palette for UIX pages looks the same and works in a similar way as the Data Control Palette for JSP pages. However, the contents of the *Drag and Drop As* pulldown are slightly different because different controls are available in UIX.

Most of the controls on the collection level have a parallel control in the JSP data control set so you should be able to determine what is available by browsing the pulldown. The attribute level contains many controls that correspond to UIX tags. Refer to the UIX Reference in the JDeveloper help system for descriptions of these controls. On the collection level, master collections (such as DepartmentsView1 used in a master-detail form) have different controls from detail collections (such as EmployeesView3, which is linked to DepartmentsView1).

Steps for Creating a Struts UIX Application

The following hands-on practice demonstrates how to create a set of working pages that perform basic data manipulation and querying. The following describes the general steps for creating a UIX application in JDeveloper. Many of these steps are the same as the steps used to create JSP applications.

1. Create a business services project containing the data collections required for the pages. For example, the hands-on practice that follows uses an ADF BC project.
2. Run the UIX wizard to create a file either by using a prebuilt template or by specifying the page design areas in the wizard pages. Templates provide the most reusability. Create all pages required for the application in this way. As a shortcut, the following hands-on practice copies the wizard-generated file instead of taking the time to create a reusable template.
3. Create a Struts page flow diagram containing the data actions and data pages linked by forwards. Link the UIX pages to the data pages in the diagram.
4. Drop the high-level data controls (such as Input Form or Search Form) from the Data Control Palette onto each page.
5. From the Component Palette, drop onto the page all required navigation buttons or other interface objects not in the default, high-level component.
6. Connect the navigation and action buttons and links to events and specify the data action or data page that will be called for each event.
7. Adjust properties as required using the Property Inspector.
8. Modify or define the data binding for elements if needed.
9. Test the application by selecting Run from the right-click menu on the startup file in the page flow diagram. As with the JSP runtime in JDeveloper, this starts up the Embedded OC4J Server.
10. Adjust the code and repeat the development steps.

Comparing JSP and UIX

Lightweight client interfaces such as those created in the JSP and UIX styles are beneficial when the application needs to interact with database data and present flexible user interfaces. The benefits offered by a web-based solution lie mainly in the ability for users to easily access the application from any web browser. Web-based solutions also offer the benefit of easy maintenance; you do not need to install or refresh the installation on user's machines. All application deployment is centralized and highly controllable. The runtime deployed code runs in different servlets for each alternative, but the basic runtime requirements—a web server with a web tier container process—are the same.

If you choose to use a lightweight client solution, you may need some guidelines about whether to use UIX or JSP. The following discussion outlines some decision points that you can use and assumes that you will use JDeveloper 10g to develop the code. (Development of UIX code in any other tool is not nearly as easy as it is in JDeveloper.)

When To Use JSP Technology

Although you can accomplish the task of building a lightweight client, web-based application using either JSP pages or UIX technology, here are some reasons that you might want to choose JSP over UIX:

- **You have existing applications using JSP pages** and do not want to learn a new technology. Although developing UIX pages in JDeveloper requires fundamentally the same set of steps, the details of how you manipulate the code are different. UIX specifics, as with any new language and technology takes some time to learn.
- **You have pre-existing JSP templates or a JSP look and feel.** One of the biggest benefits of using UIX technology is that templates are inherent in the architecture. Templates in JSP technology are an add-in. Therefore, a standard look and feel is easier to accomplish using UIX, but many shops have developed JSP templates that serve the same purpose as the UIX common look-and-feel options.
- **You do not mind designing a look and feel.** Even if you do not have existing JSP templates, you may have expertise in house to develop them and can take some time to do so at the start of the project.
- **You need industry-wide vendor support.** Although UIX is J2EE-compliant and can be thought of as a specialized code library for J2EE web development, support is limited to Oracle corporation and its consulting partners. There is a small number, if any, third-party vendors using UIX. JSP technology, on the other hand, is used by, and supported by many third-party vendors (one of which is Oracle).
- **You need assistance from a large user community.** JSP pages have been available to the user community for many years and the knowledgebase of developers who have used it is very large. You will be able to find many solutions, tips, and techniques for JSP pages by doing a simple web search. In addition, JSP page development is not limited to Oracle-oriented developers. It crosses many platforms which makes the user community support even larger.
- **You have sufficient in-house Java expertise.** JSP pages require more Java coding to customize controls and accomplish the complex behavior that is native to UIX. If your development team includes Java experts who can manipulate and extend the base behavior of JSP pages, you will be able to code complex interface requirements. Without Java skills at the expert level, this is a difficult task.
- **You need to be 100% J2EE now.** If it is important (due to political, managerial, or other reasons) for your technology choices to align closely with J2EE, JSP is a logical choice. Although UIX is J2EE-compliant and, in a future incarnation, may be closer to JavaServer Faces, another J2EE initiative, in its current version, it fits into J2EE as an extension. (See the sidebar "What is JSF?" for a brief explanation of this technology.)

What is JSF?

JavaServer Faces (JSF) is a relatively new (about a year old as of this writing) J2EE standard that defines an extension to JSP pages. It is a framework consisting of tag libraries that you can use inside JSP code. The tag libraries offer a controller for page flow, rich components (like UIX controls) that build large areas of a page in a certain way (like a scrollable table of records), and interface component events. Although this technology is new, it has gained popularity because of its ease-of-use and because of the event model.

When To Use ADF UIX

Although JSP development is supported extensively in JDeveloper, many other tools on the market allow you to easily create JSP applications. JDeveloper, on the other hand, is the best tool for creating ADF UIX applications. Just as there are several indicators that point to choosing or avoiding JSP applications, there are some strong reasons you might want to choose ADF UIX as outlined here:

- **You need to extend or supplement Oracle Applications.** As mentioned, the E-Business Suite uses UIX as the language for its self-service, web applications. If you are writing your own extensions to these applications, or if a majority of your user interfaces use the E-Business Suite applications, UIX is a logical choice for extensions or additions that you write. Since UIX incorporates the Oracle browser look and feel (BLAF) as its main template set, the applications you create using UIX also have an appearance and behavior similar to the prepackaged applications. You can also use an add-in to JDeveloper 9i called *Oracle Application Framework* (OAF), available as of release 11.5.10, to assist with development of UIX extensions to the E-Business Suite. This add-in is not available yet for JDeveloper 10g.

- **Your shop consists mainly of "traditional" Oracle developers** who are not completely in tune with JSP and Java technology. UIX offers much "drop in" functionality somewhat akin to the rapid application development (RAD) capabilities of Oracle Forms Developer. Low-level UIX or Java coding is not as frequently required because much default functionality is built into the UIX components. In addition, you can use *JHeadstart*, another extension to JDeveloper, to generate a startup application that you can customize. (The sidebar "What is JHeadstart?" explains this product a bit more.)
- **You want a pre-built look and feel.** If you do not want to spend significant time designing the look and feel of the application, you can customize the UIX templates and page layout areas fairly simply and quickly. The resulting template offers a tested, user-friendly interface that will serve many requirements.
- **You want to take advantage of the rich UIX component set.** For example, instead of coding a handful of JSP HTML controls, you can drag a single control onto the page for a search page with multiple OR search criteria. This type of component is available by default in UIX.
- **You can use Oracle support and Oracle-centric user communities for assistance.** Since UIX is an Oracle-specific technology, support for problems and questions you may have must come from Oracle itself or user communities that are oriented specifically towards UIX (mainly the OTN forums). As mentioned before, vendor support outside of Oracle is nearly non-existent although some consulting firms specialize in UIX work.
- **You do not mind a non-native J2EE technology.** UIX is J2EE-compliant and works with the design patterns and architectures of J2EE but it is not native to the J2EE specifications (JSP is native). This is not a deal-breaker for most shops because they have used Oracle Forms and other non-J2EE technologies exclusively for many years. In addition, the production release of JDeveloper 10.1.3 will incorporate JSF (introduced earlier) which is a native J2EE technology. Knowledge and code in UIX will transition to JSF knowledge and code because their foundations are similar (rich components with event models).

What is JHeadstart?

JHeadstart is an extra-cost add-in to JDeveloper 10g that was created by Oracle Consulting (www.oracle.com/ technology/consulting/ 9iServices/ JHeadstart.html). You can use it to migrate application modules and database schema elements from Oracle Designer to J2EE UIX code. In addition, it contains a standalone application generator that you can use even if you do not use Designer. This generator creates fully functional UIX pages with basic page flow navigation, master-detail, DML, search, and browse capabilities. For most application design requirements, you will modify the code that JHeadstart generates but the generated functionality saves a lot of time and effort.

Conclusion

This white paper has given an overview of the MVC design pattern and two alternatives for creating the view layer using JDeveloper—JavaServer Pages technology and ADF UIX. It explained the architecture and development steps for each of these alternatives as well as described the benefits and drawbacks of each. With this information, you should be able to begin making decisions about which view layer alternative is best for your situation. Whichever alternative you choose, JDeveloper 10g can assist greatly in the development and deployment of the code.

About the Author

Peter Koletzke is a technical director and principal instructor for the Enterprise e-Commerce Solutions practice at Quovera, in Mountain View, California, and has 20 years of industry experience. Peter has presented at various Oracle users group conferences more than 130 times and has won awards such as Pinnacle Publishing's Technical Achievement, Oracle Development Tools Users Group (ODTUG) Editor's Choice, ECO/SEOUC Oracle Designer Award, and the ODTUG Volunteer of the Year. He is an Oracle Certified Master and coauthor, with Dr. Paul Dorsey, of the Oracle Press (McGraw-Hill Osborne) books: *Oracle JDeveloper 10g Handbook* (from which some of the material in this paper is taken) and *Oracle9i*

JDeveloper Handbook (also co-authored with Avrom Roy-Faderman), *Oracle JDeveloper 3 Handbook*, *Oracle Developer Advanced Forms and Reports*, *Oracle Designer Handbook, 2nd Edition*, and *Oracle Designer/2000 Handbook*. ourworld.compuserve.com/homepages/Peter_Koletzke.

Quovera is a business consulting and technology integration firm that specializes in delivering solutions to the high technology, telecommunications, semiconductor, manufacturing, software and services, public sector and financial services industries. Quovera deploys solutions that deliver optimized business processes quickly and economically, driving increased productivity and improved operational efficiency. Founded in 1995, the company has a track record of delivering hundreds of strategy, design, and implementation projects to over 250 Fortune 2000 and high growth middle market companies. Quovera's client list includes notable companies such as Cisco Systems, ON Semiconductor, New York State, Sun Microsystems, Seagate, Toyota, Fujitsu, Visa, and Cendant. www.quovera.com.