CHAPTER 3

Creating BC4J Applications with the IDE Tools

Programmers are tools for converting caffeine into code.

—Unattributed, from a compilation of
humorous computer quotes by Guillaume Dargaud

he JDeveloper Integrated Development Environment (IDE) tools can be used to create applications without relying on the wizards. At times this is necessary because the wizards do not always produce the desired results. In this chapter you will build three projects: a BC4J project, a Java application, and a JSP application. These projects are similar to projects you create with the wizards in Chapter 1. Part II contains more detailed information about working with BC4J projects.

The goal of this chapter is to provide you with a feel for the kinds of activities required and a sense of how long the process of creating a BC4J application will take, but will not necessarily give you a thorough understanding of how to build an application.

You may notice that many of the activities in these practices require significantly more work in JDeveloper than in products such as Oracle Developer or PowerBuilder. Because of the BC4J components, it is probably not as much work as building similar applications in C++ or Visual Basic; however, it is certainly more labor intensive to build simple applications using JDeveloper than it is using Oracle Developer. The justification for the additional effort is that JDeveloper is so much more flexible and fully featured. Be aware that the JDeveloper environment is somewhat more complex than that of traditional Oracle client/server development tools.

Working with JDeveloper means working with Java in a 3GL environment. However, the JDeveloper IDE and wizards allow you to write 3GL code using a 4GL-like environment. Because JDeveloper allows you to "go behind" the code generator and write your own Java code, your applications will not suffer from any of the functional limitations that 4GL applications usually have. Such limitations do not exist in building Java applications. Virtually anything you can imagine can be supported with a proportionate amount of effort. However, the underlying business logic is still very flexible. As with any other tool, once you become familiar with the JDeveloper IDE, learning to use the features of JDeveloper will be straightforward.

About the BC4J Project

In the hands-on practice in Chapter 1, you build a basic BC4J Java application using JDeveloper's powerful high-level wizards. In Chapter 22, you build a basic BC4J JavaServer Page (JSP) application also using the wizards. With just a few clicks in a wizard, you are able to build a functioning master-detail application. If you try to generate a BC4J project using the high-level wizards, your project may not be exactly what you want. For example, some foreign key constraints between the tables in your database may not be relevant in a particular application. You may need to go back and modify an existing application by adding or modifying a component.

Another problem is that the JDeveloper default form generator only generates single master-detail relationships. Anything like a master-detail-detail relationship will require some hand modification.

CAUTION

The renaming of a BC4J object requires modifications to be made in many places within the application. There have been intermittent problems with renaming objects. Care should be taken when doing this, particularly with BC4J components. If the application starts behaving erratically or generating unexplainable error messages, one thing to try is to delete any renamed BC4J components and re-create them with the correct names. This may not always work cleanly because the files you are deleting may be related to other files.

For normal production use of JDeveloper, you may start with the high-level wizards for the first few application objects. However, you may soon find that the wizards have limited functionality and cannot generate the code required by your application. Consequently, you will need to add components manually, integrating them to components generated by the wizards.

About the Java Application Project

In this practice, you will create a simple master-detail application using the Location and Department tables from the HR sample database. (Detailed instructions about how to set up the HR schema are discussed in the Introduction.)

In the hands-on practice in Chapter 1, you created a workspace, LocDeptWS.jws. You used the wizards to create this workspace and the two projects it contains. In this chapter, you will be building the same application as in Chapter 1 without using the wizards in a second workspace, LocDept2WS.jws.

The LocDeptBC4J project supports the connection to the database, and the LocDeptJA application project includes the user interface components. Within the BC4J project, you will see its association to the connection in the Structure window after clicking the jpx file under the BC4J project. A single package—locdept—holds all of the BC4J objects. The LocDept application holds the client data model (LocDeptJA.cpx) and four classes. This application allows you to look at the Department and Location information in the HR schema included with Oracle9i JDeveloper.

Using the high-level wizards is not the only way to build the application. In this practice, you will create a functionally equivalent application by building all of the structures for the LocDept application using lower-level wizards. As you build the application, you can see what additional steps you take, and what additional flexibility these steps add to the application. For example, although the original application included four classes, this one will be structured differently. Deciding how much information should go into a class is like deciding how many procedures to use in coding. Everything may be placed in one large procedure or divided into multiple procedures that call each other.

The JClient Form Wizard built the application using four classes. For those familiar with JDeveloper 3.2, you may remember that using the wizards to generate the same application created only two classes (a frame class and an application class to call it). Deciding how to divide your code into classes is a question of style and may affect performance.

To be able to periodically examine your application and to compare your progress to what was generated by the JDeveloper wizards, this practice will stay as close as possible to the way that the wizards generate code.

About the JSP Project

For web-based applications, most developers will want to build a JSP. In this chapter, you will get a taste of how easy it is to build JSP pages. You will find that many of the development techniques are quite similar to the ones you used to build the other projects, but there are some significant differences. You should pay particular attention to how data tags are built and used.

You will be able to use the same BC4J project as you did for the Java application. This will help to demonstrate the power and flexibility of BC4J.

Hands-on Practice: Create a BC4J Project Manually

The goal of this practice is to build an application using techniques that are akin to those required for building production applications, but kept at a simple level so as not to overwhelm you with too much complexity.

For this practice, you will build the same application that you created in Chapter 1. However, this time, only the lower-level wizards will be used. The rest of the application will be built manually, using the UI Editor.

This practice consists of the following phases:

I. **Create a workspace, project, and connection**
   - Prepare a new workspace
   - Create a project for the business components
   - Create or select a connection

II. **Create entity objects and view objects**

III. **Delete and create an association**

IV. **Create a view link**

V. **Create an application module**

VI. **Test the application**

In this book, references to building applications "by hand" without using the wizards do not mean writing hundreds of lines of Java code. As mentioned in Chapter 1, there are many different wizards within JDeveloper. Some of these can be termed "high level" and enable you to build whole applications or substantial portions of applications with a few mouse clicks and entered properties. Other "low-level" wizards can be employed to create or modify a single program element. These wizards allow you to quickly build components as in a 4GL environment without having to write the

necessary 3GL code. The high-level wizards will give you a jumpstart on building your application but the low-level object wizards generate parts of the code.

One of the strengths of JDeveloper is that experienced programmers can edit the Java code directly for any application components. Even the XML generated from BC4J can be edited. Advanced developers have found that the ability to manipulate the BC4J code is a very powerful and desirable feature. However, for most applications, the BC4J wizards are powerful enough that you will not need to manually edit the XML.

I. Create a Workspace, Project, and Connection

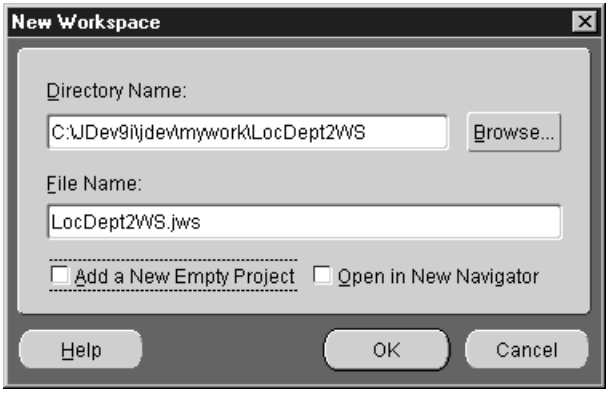
In this phase, you will create a workspace and project and connect to the HR sample schema.

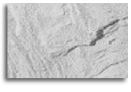
Prepare a New Workspace

To create a workspace, use the following steps:

1. On the Workspaces node, select New Workspace from the right-click menu.
2. In the New Workspace dialog, change the default name (such as "Workspace1") in the *Directory Name* field, to "LocDept2WS."

   Additional Information: The workspace directory was created as a subdirectory of the JDeveloper "mywork" folder. This workspace could be placed in any directory, but for all of the hands-on practices in this book, the mywork directory will be used.
3. In the *File Name* field, change the default workspace name to "LocDept2WS" as shown here. JDeveloper will automatically add the appropriate file extensions.





NOTE
Chapter 5 contains detailed information about the naming conventions used in the hands-on practices for this book.

4. Uncheck the *Add a New Empty Project* checkbox and leave the *Open in New Navigator* checkbox unchecked. Click OK.

   Additional Information: You will see a new workspace displayed in the System Navigator. Notice that it is shown in italics. This means that it has not yet been saved.
5. Click Save All.

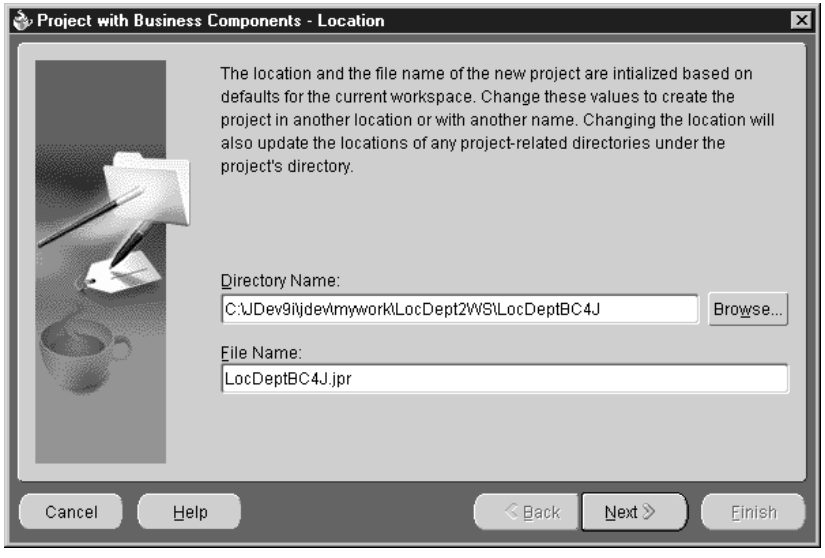
TIP

Running a project in JDeveloper will also automatically compile it. This behavior is set by the Make Project *checkbox in the Configurations\Development\Runner\Options page of the Project Settings dialog (**Project | Default Project Settings** for all future projects or **Project | Project Settings** for a specific project). Compiling a file or project automatically saves it. This behavior is controlled in the Environment page of the Preferences dialog (**Tools | Preferences**).*

Create a Project for the Business Components

Now you need to create the BC4J project within the LocDept2WS workspace.

1. In the Navigator on the LocDept2WS node, select New Project from the right-click menu. Select "Project Containing New Business Components" from the Projects category. Click OK.
2. Click Next if the Welcome page appears. On the Location page, in the *Directory Name* field, change the default name (such as "Project1") to "LocDeptBC4J." In the *File Name* field, type "LocDeptBC4J" as shown here. Click Next.



Additional Information: The BC4J project directory is a subdirectory under the LocDept2WS workspace directory.

3. On the Paths page, change the name of the *Default Package* to "locdept."

Additional Information: *Packages* refer to the subdirectory where your code will be stored. Large applications may require partitioning into several subdirectories. However,

when doing this, if a class residing in one package (directory) needs to access a class in a different package (directory), you will need to explicitly reference that class using the notation "directory name.class name." To simplify the hands-on practices, the same package name will be used throughout.

The *Java Source Path* is a subdirectory under the package directory to store the source code. The *Output Directory* will store the compiled code. Both of these subdirectories sit conveniently underneath the BC4J project directory.

NOTE
The directory structure is a change in architecture from JDeveloper 3.2. In v. 3.2, JDeveloper took the hierarchical structure of your entire application and duplicated it, storing the generated classes in one structure and the source code in another. This new architecture is much more convenient and simplifies deployment. Chapter 1 explains this directory structure further.

4. Click Next and Finish to complete the Project Wizard and to display the Business Components Package Wizard. Click Next if the Welcome page appears.

NOTE
The first time you enter any wizard, you will see a Welcome page. If you do not want to see this page each time you enter the wizard, uncheck the Display this page next time *checkbox.*

Create or Select a Connection

If you haven't already done so, you will need to create a connection for your project. (See Chapter 1 for details about creating a connection to the HR schema.)

1. On the Package Name page of the Business Components Project Wizard, ensure that the *Package Name* field shows "locdept." Leave the default radio group selection. Click Next.
2. Ensure that "HR" appears in the *Connection Name* field and click Next.
3. Click Finish. Normally, you would click Next to fill out additional properties of the BC4J package but this practice will show you how to build the components by hand.
4. Click Save All.

What Just Happened? You created a new workspace directory, BC4J project directory with a package, and database connection for the BC4J project. These structures serve as logical containers for the files that you will create in this practice.

If the application you built by hand in this practice does not work, you may be able to find errors by comparing your application to the one generated by the wizards that you created in the Chapter 1 hands-on practice. If necessary, compare the files in the Navigator for the two projects.

II. Create Entity Objects and View Objects

In this phase you will add business components to the project that you created in Phase I. Note that the Navigator window now displays an association to the connection in the LocDeptBC4J.jpx file. If you double click the .jpx file, the Business Components Project Wizard will open, and you can change the connection for your project.

1. In the System Navigator, on the LocDeptBC4J.jpr project node, select New from the right-click menu.

TIP

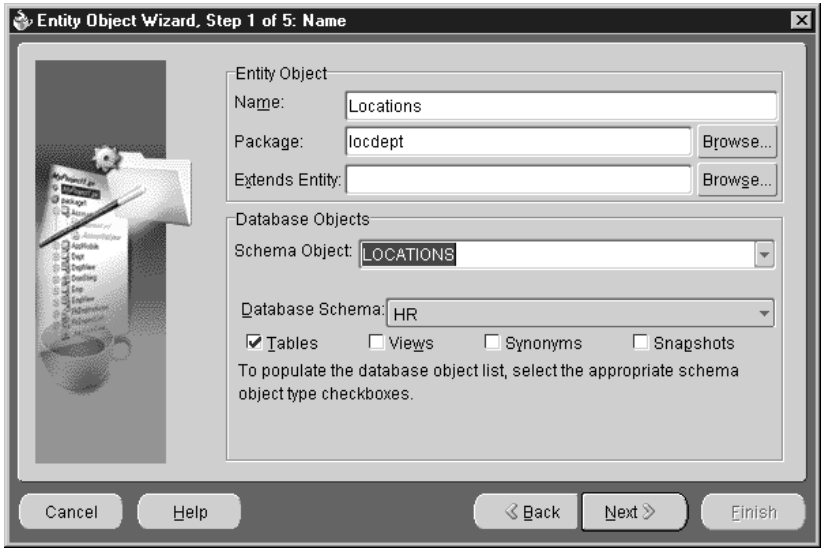
This "New" gallery will be used often in building applications with JDeveloper. It can be accessed by right clicking a workspace or project in the Navigator, selecting ***File*** *|* ***New*** *from the pulldown menu, or pressing* CTRL-N *on the keyboard. Some developers find the keypresses more efficient.*

2. Expand the Business Tier and select Business Components (BC4J). Select Entity Object from the Items list. Click OK. The Entity Object Wizard will open.

   Additional Information: Until you have associated the BC4J project with a connection, you cannot select any other object in this area. If you are unable to select an Entity Object, it is because you have not created the appropriate association to a connection. This is important since it is a common error to select Finish too soon when creating the BC4J project. If you do, only the Business Components will be available in this dialog. If this is the case, select Business Components Package and follow the wizard to the same point where the connection has been defined for this project.

3. Click Next if the Welcome Page appears to display the Name page.

4. On the Name page, select "LOCATIONS" from the *Schema Object* pulldown. This will automatically populate the *Name* field with "Locations" as shown here. Click Next.



5. On the Attributes page, all of the entity attributes derived from the LOCATIONS table columns will be displayed. You may change the order of the attributes, add a new attribute, or a remove an attribute. For now, just click Next.

   Additional Information: Changing the order or the available attributes in the entity object has little effect on the running of the application. The application will be based upon a view object that has an entity object at its core, but the view object can have a different attribute list or attribute order. Physically, the order of the attributes in the generated code will be changed as well as their appearance order in the subsequent steps. In general, when building applications, unless the table has a very large number of attributes, it is common practice to include all of them whether or not they will ultimately be deployed.

CAUTION

If you remove attributes here, they will not exist in the BC4J layer and will not be available in your application. However, if you remove an attribute, you can always re-enter the Entity Object Wizard and add the attribute back.

6. On the Attribute Settings page, ensure that the *Persistent*, *Primary Key*, *Mandatory,* and *Queriable* checkboxes are all checked for the LocationId attribute. Accept all of the default attribute settings.

   Additional Information: The *Primary Key* and *Mandatory* checkbox information is retrieved from the database when the project is run. When checked, the *Mandatory* option causes the enforcement of a NOT NULL constraint in the BC4J layer, which does not require any database access.

   The Entity Object Wizard automatically makes changes to the attribute names. For example, the database column name LOCATION_ID is transformed to LocationId. Select each attribute using the pulldown on the *Select Attribute* field, and observe the checkbox settings used by default.

7. Click Next. Accept all of the default settings on the Java page. Click the Help button for more information about the options on this page. Click Next.

8. For simple applications, the default view object is adequate, and you can create this view object by checking the *Generate Default View Object* checkbox on the Generate page. However, in order to practice creating the view object outside of the wizard, uncheck the *Generate Default View Object* checkbox. Click Next and Finish to create the Locations entity object. Additional information about view objects can be found in Chapters 10 and 13.

9. To create the default view object, expand the LocDeptBC4J.jpr and locdept nodes. On the Locations node, select New Default View Object from the right-click menu.

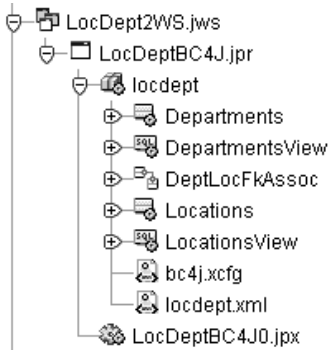
10. Click Save All.

11. Repeat steps 1–10 in this section to add an entity object and associated view for the DEPARTMENTS table, selecting "DEPARTMENTS" from the *Schema Object* field pulldown.

Additional Information: When you are finished, under the LocDeptBC4J project, you should have created new business components: an entity object for the Departments table and a corresponding view object. There will also be an association called "DeptLocFkAssoc."

12. To save your work, click the Save All icon.

At this point, your Object Navigator should look something like this (you may or may not see a bc4j.xcfg file):



What Just Happened? By creating Location and Department BC4J entities, BC4J also automatically generates an association between them based on the foreign key constraint in the database. All of the necessary Java classes and XML files are created to support the Java application's interaction with the database.

In this phase, you also generated default BC4J view objects that correspond to the BC4J entity objects. View objects are the application interface layer with which your UI components will interact.

III. Delete and Create an Association

When you created the Departments entity, JDeveloper automatically generated an association to Locations from the foreign key constraint in the database. If necessary, you can delete the automatically generated associations and create the desired associations by hand.

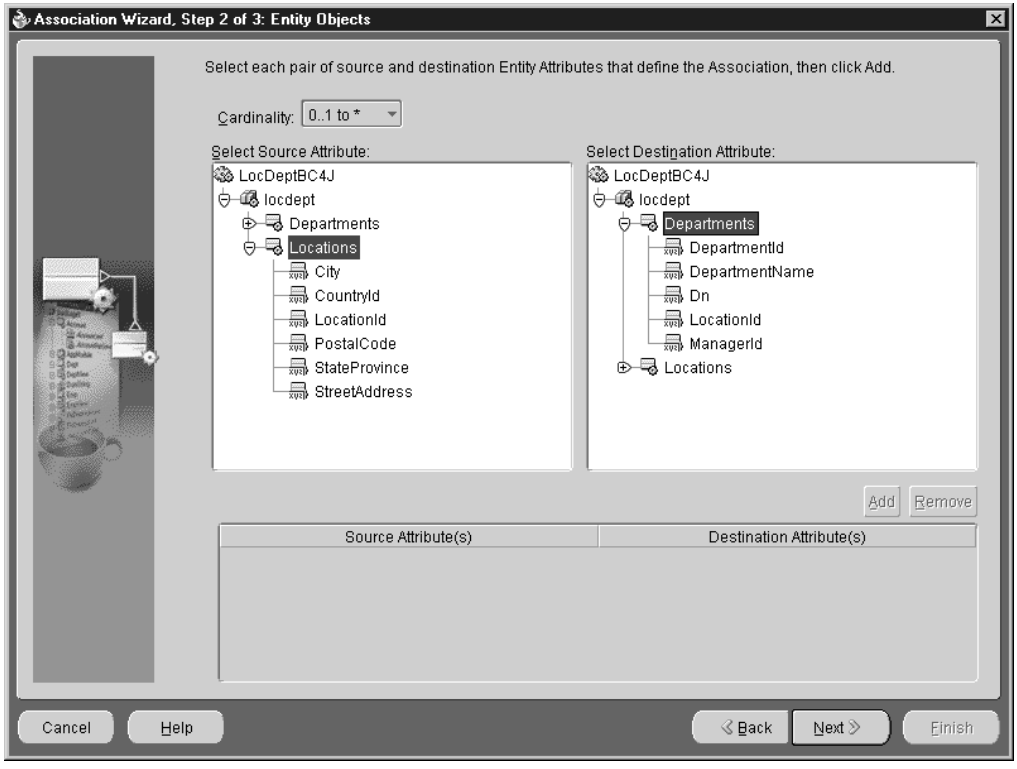
Associations can be identified in the System Navigator by an icon with small red arrows between two small squares. If you hold the mouse cursor above the icon, a tooltip will appear that identifies the object type. For this simple example, the generated associations are correct. However, to practice this operation, you will delete a generated association and create it manually. You would have to create an association manually if no corresponding constraint exists in the database.

1. To delete the generated association, select the DeptLocFkAssoc node, and then select Erase DeptLocFkAssoc from disk from the right-click menu. Click Yes on the confirmation dialog.
2. To create an association between the Departments and Locations entities, click either the Departments or the Locations node within the LocDeptBC4J.jpr project. Select New

Association from the right-click menu. Click Next to advance to the Name page if the Welcome page appears.

3. On the Name page, change the name to "LocDeptFkAssoc." Ensure that "locdept" appears in the *Package* field. Click Next.

4. On the Entity Objects page, expand the Locations node under *Select Source Attribute* on the left and the Departments node under *Select Destination Attribute* on the right as shown here.



Additional Information: The source object represents the master in a master-detail relationship and the destination represents the detail.

5. Note that the attribute names on this page are the BC4J entity attribute names and not the column names. Select LocationId under Locations on the left and LocationId under Departments on the right. Click Add. You will see the Source and Destination Attributes added at the bottom of the Entity Objects page.

6. Ensure that the *Cardinality* pulldown is set to "0..1 to *". Change it if necessary.

   Additional Information: The cardinality settings of "0..1 to *" are usually displayed as the default cardinalities. If the column is mandatory, the cardinality will still be "0..1 to *" (rather than "1 to *" as you would expect).

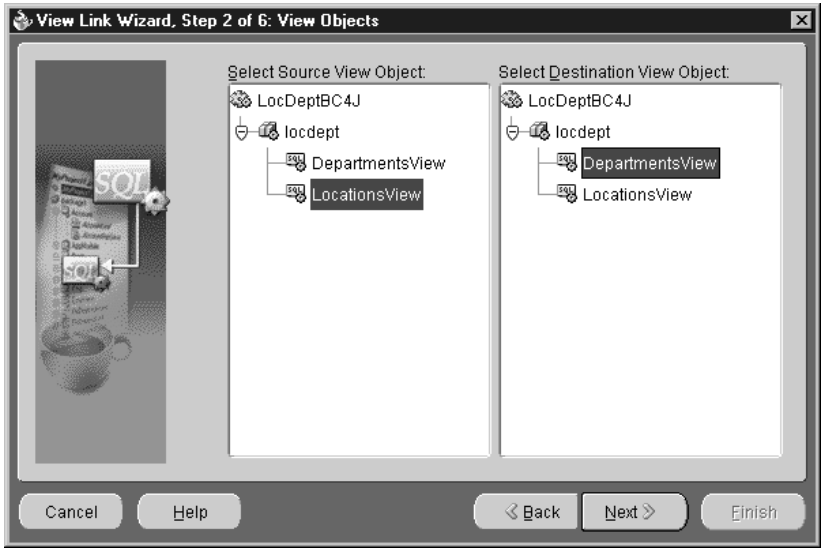
7. Leave the default settings on the Association Properties page.
8. Click Next. Examine the Summary page to see that the appropriate association object will be generated. Make alterations if needed by clicking the Back button to return to the appropriate page. Click Finish.
9. Click Save All.

What Just Happened? In this phase, you created associations between the Departments and Locations entities. You will often need to insert, modify, or delete BC4J associations. If your database does not have the requisite foreign key constraints, the appropriate BC4J associations can be created using the wizard, as described in this phase.

IV. Create a View Link

The next step is to create a view link for the view objects to correspond to the LocDeptFkAssoc association for the entities.

1. On the LocDeptFkAssoc node, select New View Link from the right-click menu to access the View Link Wizard. Click Next if the Welcome page appears to display the Name page.
2. On the Name page, change the *Name* to "LocDeptFkLink." Click Next.
3. On the View Objects page, select LocationsView in the *Select Source View Object* area and DepartmentsView in the *Select Destination View Object* area as shown here.



4. Click Next. On the Source Attributes page, select LocDeptFkAssoc and click the right-arrow button. Click Next.
5. On the Destination Attributes page, the LocationId will already appear in the *Selected Attributes* list because you selected the association on the preceding page. Click Next.
6. On the View Link SQL page, click Test to ensure that the query is valid. Click OK on the confirmation message. You can also click Explain Plan to show the explain plan information. The first time you run this in a schema, the wizard may need to create the PLAN_TABLE. Click Next.
7. On the View Link Properties page, ensure that the *Source Cardinality* is set to "0..1," and that the *Destination Cardinality* is set to "*." Accept the other default selections. Click Next.
8. The Summary page will display a structure similar to the Summary page for the association created previously. Click Finish to create the association.
9. Click Save All.

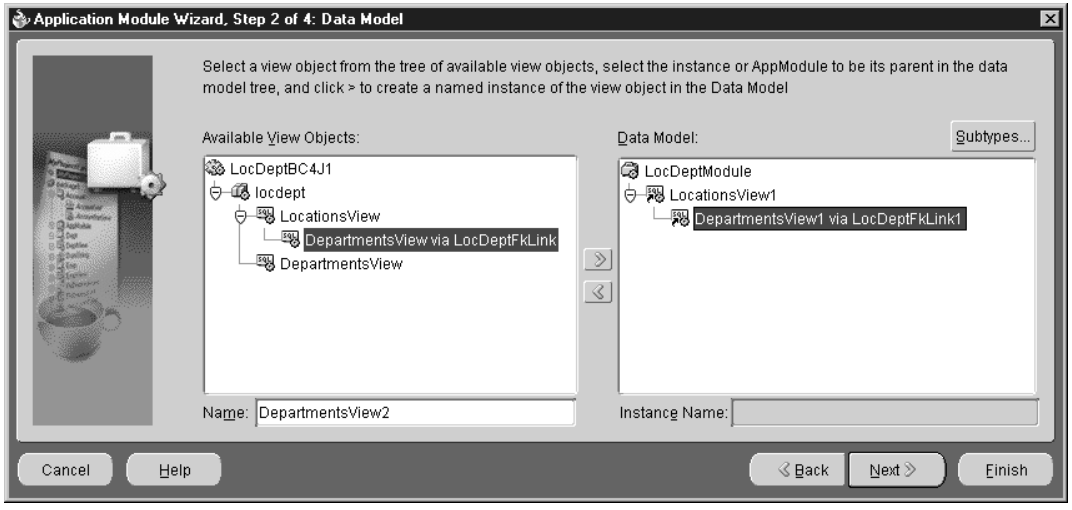
What Just Happened? In this phase, you deleted and created a view link between the Locations and Departments view objects. A master-detail relationship was created between the two view objects to enable you to build an application that supports the master-detail relationship.

V. Create an Application Module

At this point, you must create an application module for the BC4J project. The application module provides a connection point for the application to the BC4J view objects and view links.

1. On the locdept package node in the System Navigator, select New Application Module from the right-click menu to start the Application Module Wizard. Click Next if the Welcome page appears to display the Name page.
2. Type "LocDeptModule" in the *Name* field. Click Next.
3. On the Data Model page, since departments are always maintained within the context of their location, you are going to define the LocationsView as the parent and DepartmentsView as its child. To do this, select LocationsView under *Available Objects* and click the right-arrow button. "LocationsView1" will appear in the Data Model pane. This represents a view usage.
4. Click the "DepartmentsView via LocDeptFkLink" (you may have to scroll right to see the whole name) under Available View Objects, and click LocationsView1 under Data Model.

5. Click the right-arrow button so that the Data Model page of the Application Module Wizard looks like this.



Additional Information: Note that you did not move the second DepartmentsView to the Data Model since it is unnecessary for this application.

6. Click Next. No actions are required on the Application Modules page. Click Next.
7. Accept the default *Generate Java File(s)* checkbox selection on the Java page and click Next.
8. The Summary page should look like Figure 3-1. Click Finish.
9. Click Save All.

What Just Happened? In this phase, you created an application module for the BC4J components. This application module is a structured collection of view objects, which is then attached to the user interface project and defines the interaction between the UI components and the BC4J layer.

VI. Test the Application

It is always important to test your application at different points in the development process. Once you have an application module defined, you can use the following steps to do this:

1. On the LocDeptModule node in the Navigator, select Test from the right-click menu.
2. On the Oracle Business Component Browser – Connect dialog, ensure that the HR connection is selected. Click Connect.

   Additional Information: If necessary, you can change your connection at this point to complete the test process.

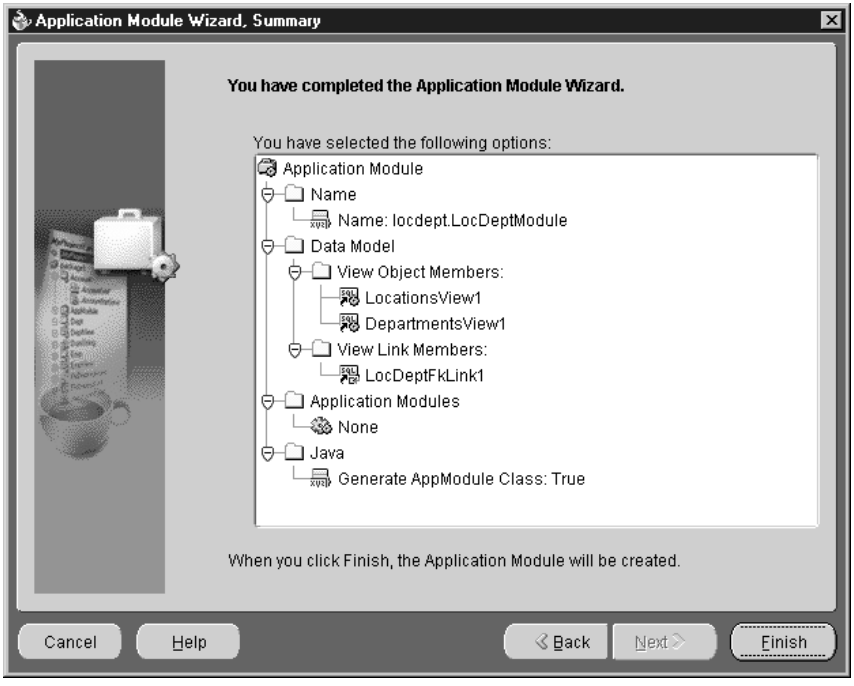


FIGURE 3-1. *Application Module Wizard—Summary page*

TIP
If you want to use the default connection, click Run in the IDE toolbar after selecting the application module node. The Business Component Browser will open without displaying the Connect dialog. Chapter 16 contains more information about connections.

3. You will now see the Oracle Business Component Browser displaying a default application based upon your selected BC4J components.
4. On the LocationsView1 node, select Show from the right-click menu.
5. Scroll through the records in the Locations table using the blue arrows in the toolbar.
6. On the LocDeptFkLink1 node, select Show on the right-click menu to display a default master-detail application as shown in Figure 3-2. Scrolling through records should display the appropriate detail departments for the selected master location.

TIP
Seattle (LocationId 1700) has many departments.

Additional Information: The correct appearance of this screen as shown in Figure 3-2 validates that the BC4J components have been built correctly. For every project you build, you should test the Insert, Update, and Delete functionality of all view objects. If you cannot perform a specific operation required by your user interface application at this point with the BC4J components, the user interface application will not function properly.

TIP

You can also double click the view object name in the Business Component Browser view objects tree to display the browser application for that view object.

7. Close the Business Component Browser window.

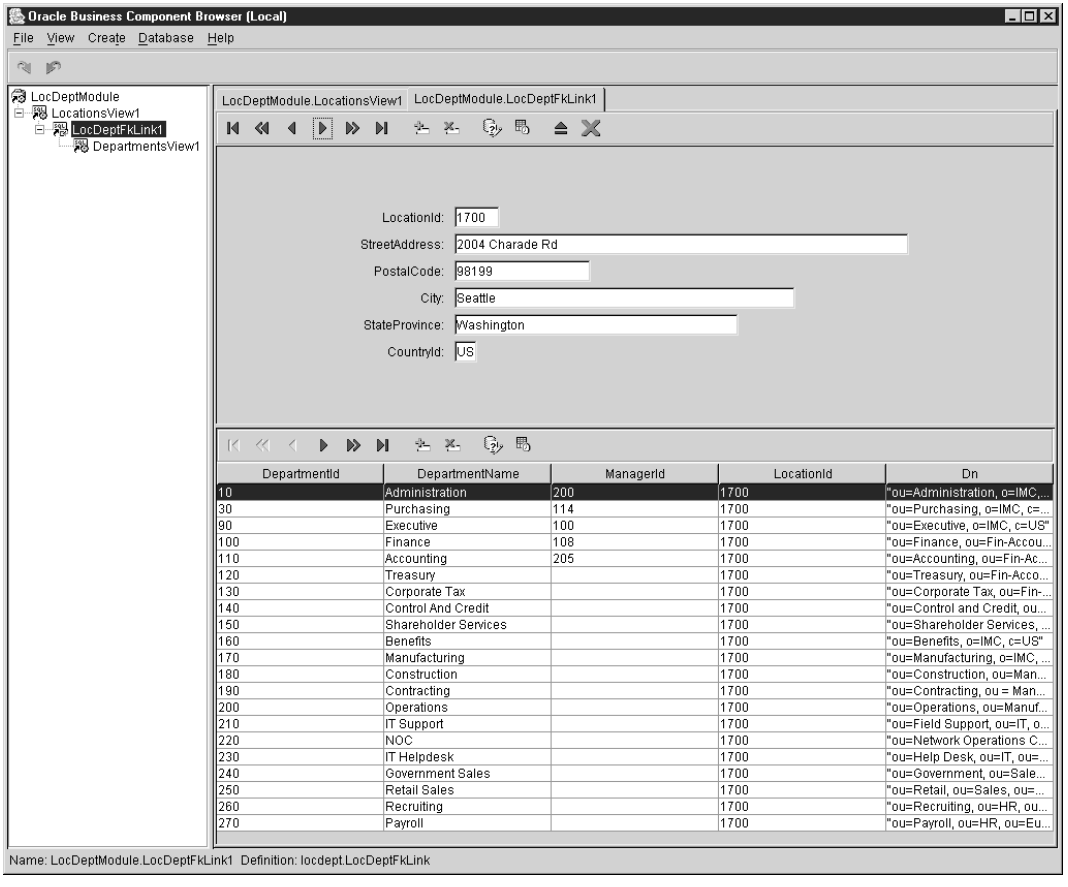


FIGURE 3-2. *Locations and Departments business components test*

What Just Happened? In this phase of this hands-on practice, you tested the application that you created to ensure that the BC4J components have been created correctly and that the Insert, Update, and Delete functionality of the application work correctly.

Hands-on Practice: Create a Master-Detail Java Application Manually

Now that you have created a BC4J project, you need to create an application that uses the BC4J components. This hands-on practice will demonstrate how to build and modify a typical application. In production application development, you will frequently start with a wizard-generated application and begin modifying it. At other times, you will need to build the application components yourself.

This practice will help to illustrate how to build manually what the wizards build automatically. It is important to understand how the wizards work. The way in which the JDeveloper wizards generate applications is not the only way to build the same applications. Even the panels and frames built here are not strictly necessary, nor are they even necessarily the way you would build an application if left to your own design. The algorithms that the wizards use have slightly changed from the 3.2 to the 9i releases of JDeveloper.

The application in this practice purposely attempts to stay very close to the way in which the JDeveloper wizards generate the application. This will be helpful in case you run into any problems during the practice since you will be able to look at the code that the wizards generated in Chapter 1 and compare it with the code that you create here.

Programming style in any language can be vigorously debated. How you perform specific tasks will evolve as your understanding of Java and JDeveloper deepens. This chapter will illustrate the way that one developer has come up with to build applications. The authors do not suggest that this is necessarily the easiest or best way, but it has been used successfully to build working applications.

This practice session consists of the following phases:

I. **Create a Java application project**

II. **Create the Locations portion of the application**

- Add a panel for Locations
- Add a JPanel container
- Add a navigation bar
- Add labels
- Add fields

III. **Create the application-level panel**

- Add an outer panel
- Modify the Java code

IV. **Test and modify the Locations portion of the application**

V. Create the Departments portion of the application

- Add a panel for Departments
- Add a JPanel container
- Add a ScrollPane
- Add a Table Component

VI. Modify the application

- Modify the size of the application window
- Change the column headings and widths

I. Create a Java Application Project

Since you will need to connect your application to the BC4J project created in the first hands-on practice in this chapter, you will use the same workspace to create the application.

Create a project within the LocDept2WS workspace using these steps.

1. In the System Navigator, on the LocDept2WS.jws node, select New Empty Project from the right-click menu.
2. In the New Project dialog, change the *Directory Name* to "LocDeptJA" (keeping the rest of the directory path intact) and the *File Name* to "LocDeptJA." Click OK.

   Additional Information: At this point, you need to associate this project with the BC4J project created in the first hands-on practice in this chapter. You must provide this visibility, or you will not be able to attach the objects in the new project to the BC4J components.
3. On the LocDeptJA.jpr node and select New from the right-click menu. Expand the Client Tier node. Select Swing/JClient for BC4J and Business Components Client Data Model under Items. Click OK.
4. In the BC4J Client Data Model Definition Wizard, click Next if the Welcome page appears to display the Definition page. Accept the defaults and click Next.
5. On the Definition Name page, JDeveloper will automatically fill in LocDeptModule. Accept this default. Click Next and Finish.

   Additional Information: You should see a LocDeptJA.cpx file under the LocDeptJA.jpr node as shown in Figure 3-3. You will only see the .cpx file in the LocDeptJA project that you just created. The .cpx file is a container that stores the *client data model,* named connections to the BC4J application module in the middle tier. You can modify this definition. The Java code will only reference the name. This allows you to redeploy or change definitions without changing any of the code.
6. Click Save All.

What Just Happened? In this phase, you created a project and associated it with the BC4J project created in the earlier hands-on practice in this chapter. The rest of this hands-on practice

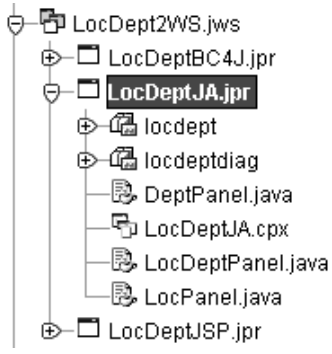


FIGURE 3-3. *Navigator showing Java application*

will demonstrate how to build a complete working application to show Departments and Locations information.

The client data model is further explained in Chapter 17.

II. Create the Locations Portion of the Application

This phase will create the Location portion of the application including a panel, scroller (ScrollPane), navigation bar, labels, and fields.

Add a Panel for Locations

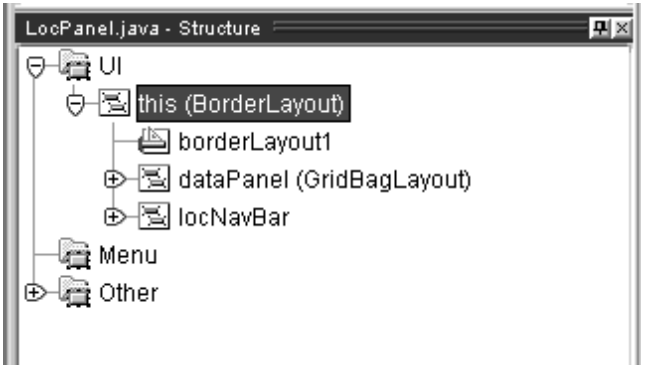
1. Click the LocDeptJA.jpr node in the Navigator and press CTRL-N. Select Empty Panel from the Swing/JClient for BC4J category. Click OK.
2. If the Welcome page appears, click Next to display the Data Model page. In the JClient Empty Panel Wizard, since there is only one data model, the data model will default to the LocDeptModule. Click Next.
3. On the "File names" page, change the package name to "locdept." Change *Panel name* to "LocPanel." Leave the *Generate a runnable panel* checkbox unchecked. Click Next and Finish. You will now see a LocPanel.java file under the LocDeptJA.jpr node and the UI Editor will open automatically.

   Additional Information: On the LocPanel.java node, select Code Editor from the right-click menu. Notice that this panel class is an implementation of the JPanel interface. JPanel is an Oracle-supplied panel that is an extension of the Swing JPanel, modified to support BC4J.

   At this point, you will add some objects to the newly created panel.

   Additional Information: Note that the Structure window on the lower left corner of the IDE displays different information when the UI Editor is active from when the

Code Editor is active. When the UI Editor is active, the Structure window shows the UI perspective of the panel object. If you expand the UI node, you will see the object "this," which refers to the LocPanel file selected in the Navigator. You will also see the name of the object at the top of the Structure window as shown here:



4. Click the "this" node in the Structure window. In the Property Inspector, set the *layout* property to BorderLayout. Layouts are explained in Chapter 20.
5. Click Save All.

Add a JPanel Container

You will now add a JPanel container to your application.

TIP

A reliable way to add objects to the Structure window is to click where the object should be added (usually under the "this" node). Then click the icon on the Component Palette for the object to be added. Finally, click back in the Structure window where you want the object to be inserted. Sometimes you will have to wait a moment before the object appears.

1. On the Component Palette pulldown, select Swing Containers.
2. Click the JPanel icon on the Component Palette. In the Structure window, click "this (Border Layout)," and you should see the new panel (jPanel1) below the borderLayout1 node.
3. Select JPanel1 in the Structure window. The following steps involve making changes in the JPanel Property Inspector.
4. On the Property Inspector, change the *name* property to "dataPanel." Note that the default *constraints* property for the data panel is Center so that the new JPanel object fills the center of the outer window.

NOTE
When filling in any information on the Property Inspector, be sure to press Enter when finished to ensure that your changes are made.

5. Set the *layout* property to "GridBagLayout."

   Additional Information: Here you have been instructed to use GridBagLayout. This is a layout where objects are placed into cells, and their position is determined by a set of properties. You may find this layout confusing until you have had some experience with it. Many developers advocate laying out their screens using the "null" layout, which allows them to position the items on the screen simply by dragging them to the correct position and then converting to grid layout at a later point in the design process. This method allows the layout conversion software to select the correct grid properties. Further discussion of the various layout managers such as GridBagLayout can be found in Chapter 20.

6. Click Save All.

Add a Navigation Bar

The following steps can be used to add a navigation bar to the application.

1. Select the "this (BorderLayout)" node in the Structure window. On the Component Palette, select JClient controls from the pulldown. Click the JUNavigationBar icon.
2. Click the "this"node to add the navigation bar object. In the UI Editor, you should see a navigation bar with grayed out icons in the center of the panel.
3. In the JUNavigationBar1 Property Inspector, set the *constraints* property to "North" and change the *name* property to "locNavBar." You must now attach the navigation bar to a BC4J component. You will see the navigation bar move to the top of the window in the UI Editor.
4. Set the *model* property to "JClient View Binding." In the model dialog, select LocationsView and click OK.
5. Click Save All. Your screen should look like Figure 3-4.

Add Labels

You now need to add labels to your application.

1. With LocPanel.java selected in the System Navigator, select Swing on the Component Palette pulldown.
2. Click the JLabel icon and click the dataPanel node in the Structure window to add the JLabel object.
3. In the JLabel1 object Property Inspector, change *name* to "locIdLabel."
4. Access the *constraints* property dialog by selecting clicking the "…" button in the *constraints* property field. Change the *constraints* dialog to match the settings in

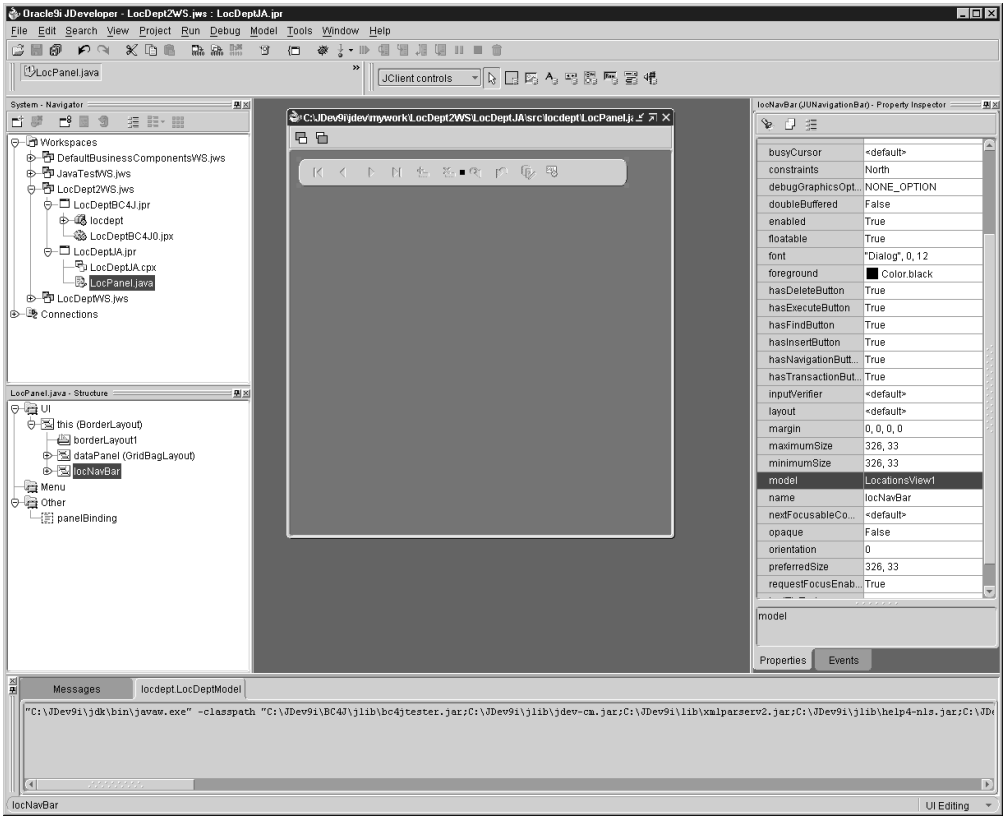


FIGURE 3-4. *Application with navigation bar*

Figure 3-5. For an explanation of the *constraints* settings, refer to the discussion of GridBagLayout in Chapter 21.

You should now see a label in the center of the panel in the UI Editor.

NOTE
If the label appears in the upper-left corner or elsewhere, you need to reset the data panel constraints property. Dismiss the constraints dialog. Select dataPanel in the Structure window, and reset the constraints property to Center.

5. In the locIdLabel Property Inspector, change the *text* property to "Loc ID."

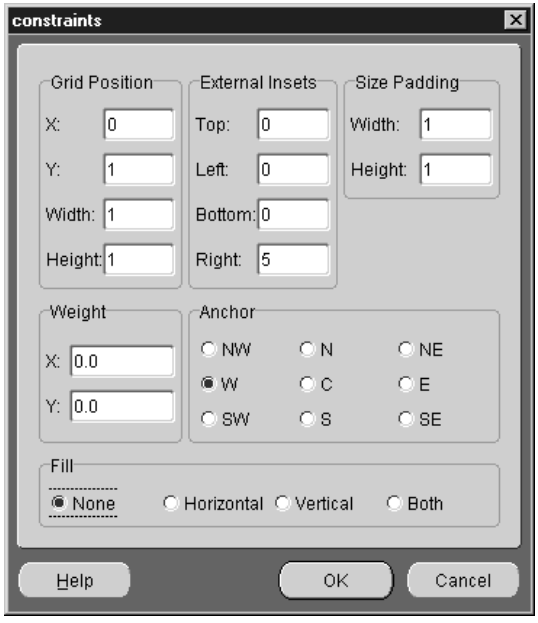


FIGURE 3-5. *Constraints property setting for the LocId label*

Additional Information: The wizard does not fill in the property name. Instead, if you look at the generated code from the wizard application, it will take the label text directly from the BC4J attribute name. Explicitly setting the text overrides the BC4J name.

6. Now, you need to add a second label for City. With LocPanel.java selected in the System Navigator, make sure that Swing is selected on the Component Palette pulldown.
7. Click the JLabel icon and click the dataPanel node in the Structure window to add the JLabel object to the dataPanel(GridBagLayout) node.
8. In the jLabel1 object Property Inspector, change *name* to "cityLabel."
9. Change the *constraints* property as shown in Figure 3-5 with the exception that Y = 2. Click OK.
10. Change the *text* property to City.
11. Click Save All.

TIP
If the Structure window does not display the UI node, click an object in the UI Editor.

Add Fields

You now need to add the associated field for the LocId label.

1. Click the JTextField icon in the Component Palette.
2. Click the dataPanel in the Structure window to add the text field as a sibling to the label.
3. In the JTextField Property Inspector, change the *constraints* settings as shown in Figure 3-5 with the following exceptions: X = 2, Y = 1.
4. Change *name* to "locIdField." The JTextField should now appear just to the right of the label in the UI Editor.
5. Select JClient Attribute Binding from the *document* property pulldown. In the document dialog that pops up, under *Select a view*, select LocationsView and under *Select an attribute*, select LocationId. Click OK.

   Additional Information: This uses the *document* property to attach (bind) the object to the BC4J component.

CAUTION

Be sure to set the document *property last since after setting the* document *property, you may see that the field shrinks down to a very narrow one. Set the columns property to "10." The* columns *property sets the width of the field. The number roughly corresponds to the number of characters to be displayed. However, you may need to adjust this number by trial and error.*

6. This same process must be repeated for the *City* field. You must create a label and field for every attribute that you want to display. Click the JTextField icon in the Component Palette.

CAUTION

You might be tempted to copy and paste the existing JLabel objects and then make the appropriate modifications. Until you are more familiar with JDeveloper, do not do this. The copy-and-paste functionality copies all of the Java code, and not all of the properties will stay linked to the generated Java code, causing unexpected application bugs.

7. Click the dataPanel in the Structure window to add the text field as a sibling to the labels. That is, both should be directly under the dataPanel node.

CAUTION

If you inadvertently place an object under the wrong structure node, do not cut and paste it. Delete it and re-add the object.

8. In the JTextField1 Property Inspector, change the *constraints* settings to match those shown in Figure 3-6 with the following exceptions: X = 2, Y = 2.
9. Change *name* to "cityField."
10. Select JClient Attribute Binding from the pulldown in the *document* property pulldown. In the document dialog select the LocationsView1 and the City attribute. Click OK.
11. Set the *columns* property to "10."
12. Click Save All.

The Structure window and UI Editor should now look like Figure 3-6.

What Just Happened? In this phase, you created the Locations portion of the application by adding panels, a navigation bar, labels, and fields. This should already give you some sense of the amount of manual effort required to build a Java application. For example, notice that labels and fields must be handled independently.

III. Create the Application-Level Panel

This panel will be used for the entire application encompassing both the Location and Department panels. The reason for creating the larger panel at this point is to allow you to test the Location portion of the application and its objects to make sure that these work before building the Department portion.

FIGURE 3-6. *UI Editor and Structure window showing labels and fields*

Add an Outer Panel

The following steps are used to create the application-level panel.

1. On the LocDeptJA.jpr node, select New from the right-click menu. Select Empty Panel from the Swing/JClient for BC4J category. Click OK.
2. If the Welcome page appears, click Next to display the Data Model page of the JClient Empty Panel Wizard. LocDeptModule should be shown in the *Select the data model definition* field. Click Next.
3. On the File names page, ensure that the *Package name* field shows "locdept" and change the Panel name to "LocDeptPanel." Check the *Generate a runnable panel* checkbox.
4. Click Next and Finish. The UI Editor will open automatically.
5. In the this (JPanel) Property Inspector, set the *layout* property to BorderLayout.
6. With LocDeptPanel.java selected in the Navigator, select Swing Containers from the Component Palette pulldown. Click the JScrollPane icon and click the "this" node in the Structure window.
7. Click the new jScrollPane1 node under the "this" node in the Structure window.
8. In the Property Inspector, change *constraints* to North and *name* to "masterScroller."
9. Click Save All.

Modify the Java Code

At this point, you must add three lines of code to the LocDeptPanel.java file.

1. Double click LocDeptPanel.java to open the Code Editor.
2. Add one line of code under the "`JScrollPane masterScroller = new JScrollPane();`" line in the BC4J binding variable section:

```
private LocPanel locPanel;
```

3. Add the following lines of code to the `jbInit()` method section under the "`this.add(masterScroller, BorderLayout.NORTH);`" line:

```
    locPanel = new LocPanel(panelBinding);
    masterScroller.getViewport().add(locPanel);
}
```

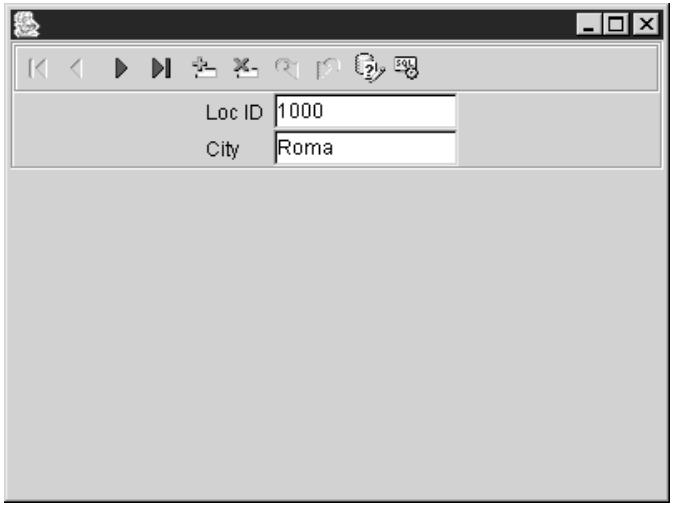
4. In the Code Editor, select Make LocDeptPanel.java from the right-click menu to compile the file and check the syntax of the code you added.
5. Click Save All.

What Just Happened? In this phase, you created a panel to hold the Location and Department portions of the application and attached the panel to the BC4J components. This gave you an introduction to working with frames and panels as well as writing code without the wizards. Note that some of the work done by hand in this phase may not be necessary as JDeveloper continues to evolve.

IV. Test and Modify the Locations Portion of the Application

At this point, the application is complete enough to run. When you generated the panel LocDept, you specified that it should create an associated application frame using the checkbox. The application frame file contains a `main()` method and is executable.

1. Select LocDeptPanel.java and click Run. The following will appear:

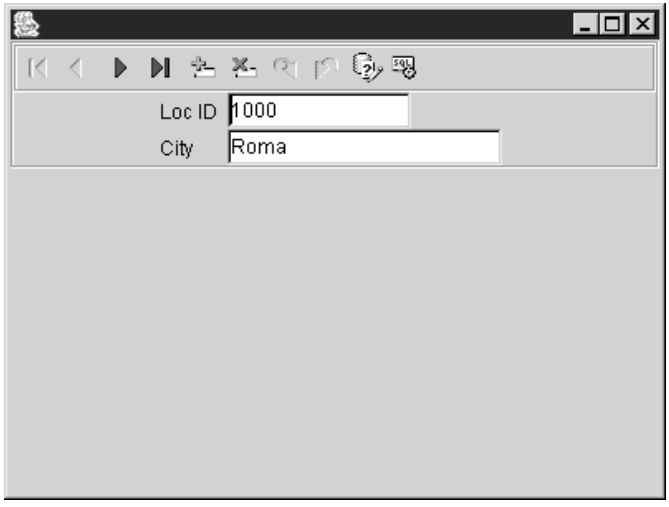


Additional Information: As you scroll through the records, you will notice that the *City* field is too narrow. You can modify this using the following steps:

2. Close the application window. On the LocPanel.java node, select UI Editor from the right-click menu. In the Structure window, expand the UI, "this," and dataPanel nodes to find the cityField object.
3. Select the cityField object and in the Property Inspector, change the *columns* property to "15."

Additional Information: This property sets the width of the field. The number roughly corresponds to the number of characters to be displayed. However, you may need to adjust this number by trial and error.

4. Run the application again to see that the *City* field has been expanded. The application should look something like this:



5. Click Save All.

What Just Happened? In this phase, you tested a portion of the application and made modifications to a field.

V. Create the Departments Portion of the Application

Now you will add the Departments portion of the application.

Add a Panel for Departments

1. On the LocDeptJA.jpr node, select New from the right-click menu. If necessary, expand the Client Tier node and select Empty Panel from the Swing/JClient for BC4J category. Click OK.
2. Click Next if the Welcome page of the JClient Empty Panel Wizard appears.
3. Click Next on the Data Model page.
4. On the "File names" page, if necessary, change the *Package name* field to "locdept." Change the *Panel name* field to "DeptPanel." Leave the *Generate a frame that runs this panel* checkbox unchecked. Click Next and Finish. You should now see a DeptPanel.java file under the LocDeptJA.jpr node.
5. In the Property Inspector, set the *layout* property for "this" to BorderLayout.
6. Expand the "this" node in the Structure window. Select BorderLayout1. In the Property Inspector, change the name to "deptLayout."
7. Click Save All.

Add a JPanel Container

To add a panel to this frame, use the following steps:

1. On the Component Palette, select Swing Containers from the pulldown.
2. Click the JPanel icon. In the Structure window, click "this," and you should see the new panel below the deptLayout node.
3. In the Structure window, make sure that JPanel1 is selected. In the Property Inspector, change the *name* property to "dataPanel."
4. Verify that the *constraints* property is set to "Center."
5. Set the *layout* property to GridBagLayout.
6. Set the *minimum size* property to "100, 100" to ensure that it is always displayed.
7. Click Save All.

Add a ScrollPane

To add a scroller to the data panel object, use the following steps:

1. Select dataPanel(GridBagLayout) in the Structure window. On the Component Palette pulldown, select Swing Containers and click the JScrollPane icon. Click dataPanel in the Structure window.
2. In the Property Inspector for the JScrollPane, change the *name* to "deptScroller."

Add a Table Component

Add a Swing JTable to the Department ScrollPane to display multiple department records using the following steps:

1. Select deptScroller in the Structure window. On the Component Palette pulldown, select Swing and click the JTable icon, and click the deptScroller node in the Structure window. You may need to resize the table in UI Editor by dragging the handles.
2. In the jTable1 Property Inspector, change *name* to "deptViewTable."
3. Set the *model* property to "JClient Attribute list binding" from the pulldown. In the model dialog, select DepartmentsView1 in the *Select a view* area. All attributes appear in the *Selected attributes* area. Move all but the DepartmentId and DepartmentName back to the left. Click OK.
4. Click Save All.
5. You will now add a ScrollPane to the panel. On the LocDeptPanel.java node, select UI Editor from the right-click menu.

NOTE
This file may already be open. Check the Document Bar in the IDE toolbar area for the LocDeptPanel.java file name and click that icon. UI Editors in this bar have an icon with a pencil and colored "page." Code Editors in this bar have an icon with a pencil and several lines of text on the "page." You can also press the ALT *key and the number key written into the icon in the Document Bar. If the Document Bar is not visible, select* ***View*** *|* ***Document Bar****. Chapter 2 contains more information about the Document Bar.*

6. Expand the UI node and select the "this" node in the Structure window. On the Component Palette pulldown, select Swing Containers and click the JScrollPane icon. Click the "this" node again, and you will see a JScrollPane added to the Structure window.
7. In the Property Inspector, change the *name* property to "detailScroller."
8. Verify that the *constraints* property is set to "Center."

   Additional Information: Notice that if you click on the masterScroller in the Structure window, its *constraints* property is set to North.
9. Click Save All.

   At this point, you need to return to the Code Editor for the LocDeptPanel.java class where the lines of code were added before. Similar lines of code must now be added here.
10. Click the Code Editor for the LocDeptPanel.java file in the Document Bar.
11. Under the BC4J binding variable comment, after the "`private JScrollPane detailScroller = new JScrollPane();`" line, add the following code:

```
private DeptPanel deptPanel;
```

12. In the `jbInit()` method, under "`locPanel = new LocPanel(panelBinding);`" insert a blank line and add the following code:

```
deptPanel = new DeptPanel(panelBinding);
```

13. Also in the `jbInit()` method, under "`masterScroller.getViewport().add(LocPanel);`" insert a blank line and add the following code:

```
detailScroller.getViewport().add(deptPanel);
```

14. In the Code Editor window, select Make LocDeptPanel.java from the right-click menu to compile the file and check the syntax.
15. Click Save All.
16. Test your application by selecting the LocDeptPanel.java node and clicking the Run icon. You should see something like Figure 3-7.
17. After checking the application, close the application window.

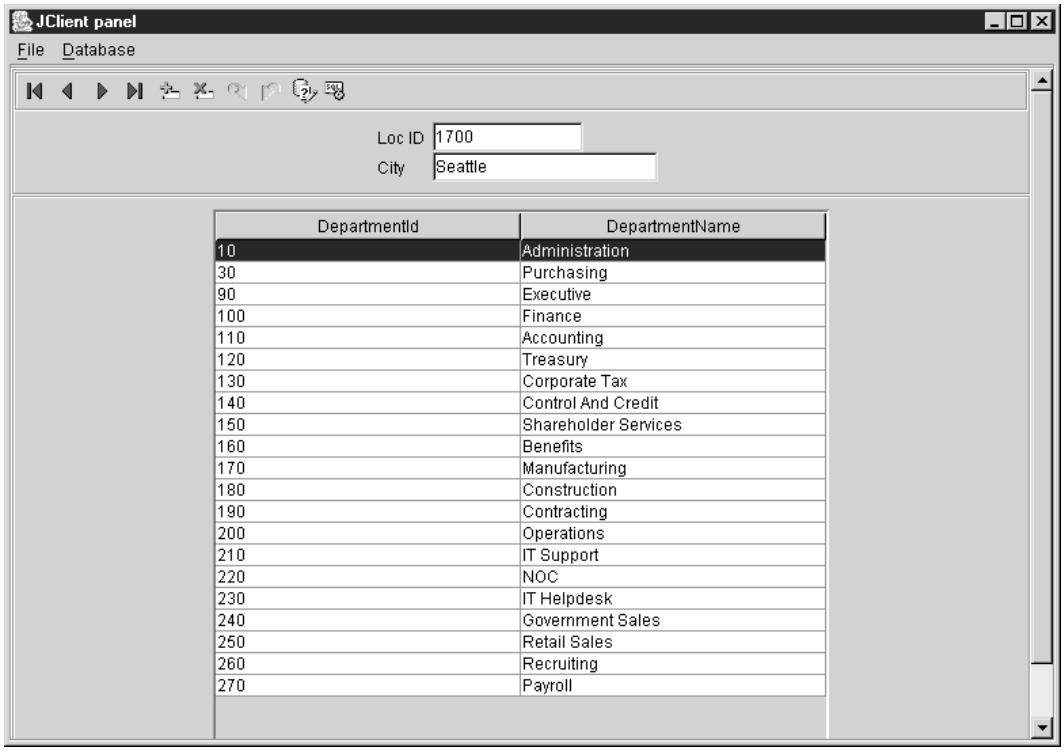


FIGURE 3-7. *Application test*

What Just Happened? In this phase, you added the Department components to the application, including panels and a ScrollPane. These are the objects necessary to show a multi-record department display as a detail to a master location record.

VI. Modify the Application

In this phase, you will test and modify the application to make it more visually appealing. In looking at the application, you may notice that there are some things that should be modified:

- The overall application window is too small.
- The column headings are not user friendly.
- The DepartmentId column is too wide.

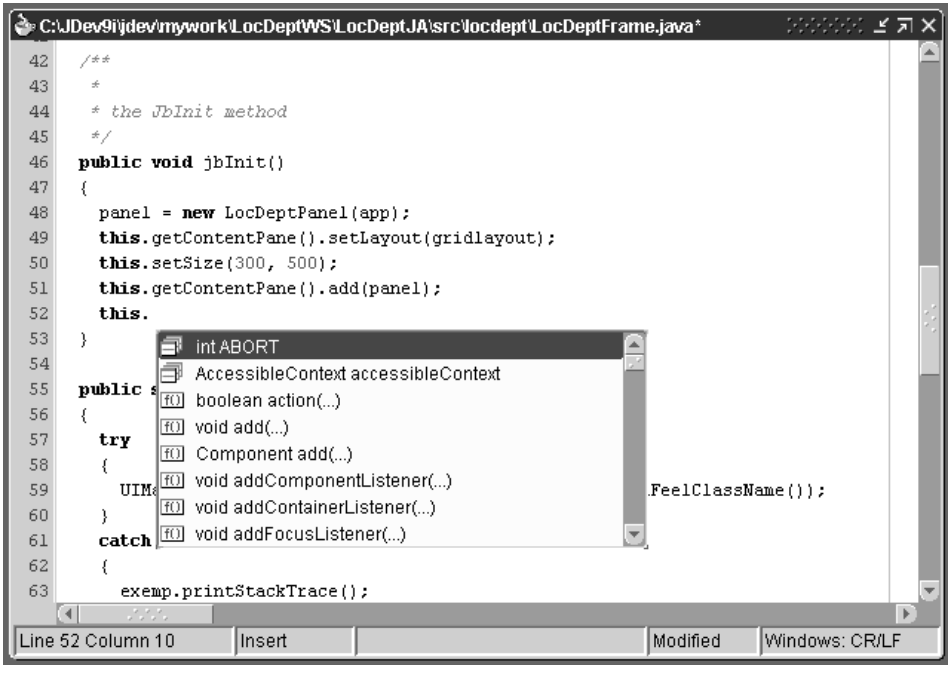
In other products such as Oracle Forms, everything you could manipulate on an object is visible in its associated Property Palette. In the Java environment, this is not the case. The JDeveloper Property Inspector is really nothing more than a limited functionality wizard

for your Swing components. A host of other things can be modified in these structures beyond what is shown in the Property Inspector. These modifications must be made in the code itself.

Some of the modifications mentioned earlier can only be made by adding code. It is strongly recommended that you purchase the official Sun reference manual so that you can understand this code (*The Java™ Class Libraries, Second Edition,* by Patrick Chan, Rosanna Lee, Doug Kramer; Addison-Wesley, 1998; ISBN: 0201310023). Be aware that as these libraries evolve, the functionality of the components will change somewhat.

TIP

To see what methods are available for an object, while in the Code Editor, on a new line in the `jbInit()` *method, type "this." (including the period). Wait a moment and you will see a popup window with a list of all the available methods for the object in question as shown here. This feature, called Code Insight, is explained further in Chapter 2.*



Modify the Size of the Application Window

Use the following steps to make the window size more compatible with the information displayed.

1. To verify that you can change the size of the application window only by modifying the code, on the LocDeptPanel.java node select UI Editor from the right-click menu. None of the properties listed in the Property Inspector will allow you to modify the application window size.

2. Double click the LocDeptPanel.java node to open the Code Editor. Find the `jbInit()` method (at approximately line 39).

   Additional Information: At approximately line 64, you will see that there is already a line of generated code:

   ```
   JUTestFrame.testJClientPanel(panel, panel.getPanelBinding(), new
   Dimension(400, 300));
   ```

   This line governs the size of the application window. Change the numbers in parentheses to "500, 300."

3. Click Save All.

4. Run the application to verify the application window size change.

Change the Column Headings and Widths

The next two modifications can be made at the same time.

1. To verify that you can change the column headings and widths only by modifying the code, on the DeptPanel.java node select UI Editor from the right-click menu. If you explore the properties, you will find that none of the properties listed in the Property Inspector will allow you to modify the headings and widths of the columns in the table component.

   Additional Information: The most likely place to do this would have been the deptViewTable object visible in the Structure window using the *model* property in the Property Inspector.

2. Double click the DeptPanel.java in the Navigator to open the Code Editor.

3. Find the `jbInit()` method by double clicking the method named in the Structure window (it is at approximately line 49) and add these four lines of code under the "`this.add(dataPanel, BorderLayout.CENTER);`" line:

   ```
   deptViewTable.getColumn("DepartmentId").setMaxWidth(60);
   deptViewTable.getColumn("DepartmentId").setHeaderValue("ID");
   deptViewTable.getColumn("DepartmentName").setMaxWidth(200);
   deptViewTable.getColumn("DepartmentName").setHeaderValue
   ("Department");
   ```

   Additional Information: Because the attribute names are within quotes, the compiler will not find an error in spelling. Therefore, it is important that the names of the attributes are spelled correctly. To avoid typing errors, you may want to copy and paste the names from the panel binding code immediately preceding.

   Unfortunately, there is no visual editor to show the appropriate column width of the JTable component, so setting the widths is somewhat of a trial-and-error process. The numbers used here, 60 and 200, can be set to whatever widths the columns should be in your application.

   The `setHeaderValue()` method takes a Java object as its parameter, so the argument passed here could be text or anything else.

4. Click Save All.
5. Run the LocDeptPanel.java again to confirm the changes. It should look something like Figure 3-8.

What Just Happened? In this last phase of the hands-on practice, you were able to modify the application you created earlier using the Code Editor and adding lines of Java code. This was necessary since there are no properties available in the Property Inspectors for the relevant objects to make these modifications.

The first two hands-on practices in this chapter used some of the basic skills needed to build applications in JDeveloper. You built a BC4J project and an application project, and made them interact. This chapter has not touched on several important other areas of application development (menus and toolbars) since these are discussed in detail in Chapter 20.

You may have come to the conclusion that building applications by hand in JDeveloper is somewhat more labor intensive than working with other 4GL products such as Oracle Forms. However, in return, JDeveloper grants much greater flexibility to the application programmer. It is a relatively common problem when developers want their tools to do something but are

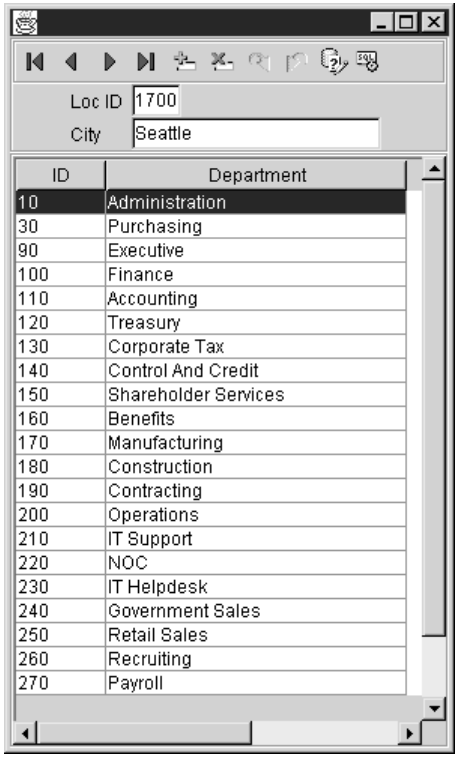


FIGURE 3-8. *Completed application*

hampered by a product's limitations. Such frustrations are far less likely with JDeveloper. However, there will still be times when you will want to make user interface decisions to minimize the amount of work required in the JDeveloper product. In every product, developers have to balance their desire for the ultimate UI with what the product can easily support.

Hands-on Practice: Create a JSP Application Manually

The BC4J, Java application, and JSP projects you created in Chapter 1 use the high-level wizards. In this practice, you will build a similar application using the low-level wizards and some hand coding. This will give you a feel for how to build JSP applications. Chapters 21-24 provide further discussion of JSP development.

This practice steps you through the following phases:

I. **Set up the project and attach the application module**

II. **Add a data source and data tags**

III. **Modify the display**

- Apply a style sheet
- Modify the label properties

This practice creates a page that displays one record at a time from the department table. It uses BC4J data tags to navigate through the records and to display records. You enter the components using the Component Palette, which prompts you for the BC4J data specifics and other properties such as names.

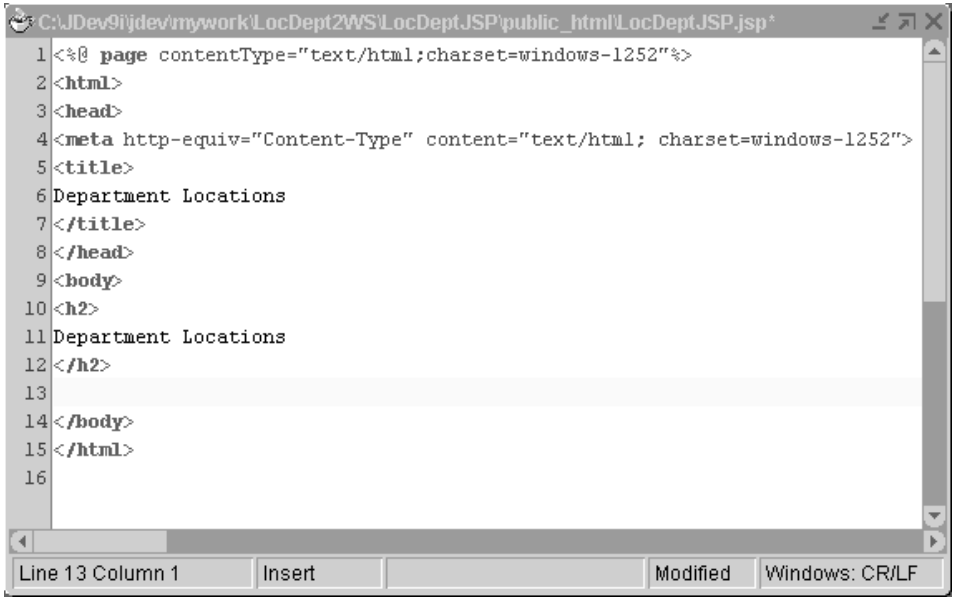
I. Set Up the Project and Attach the Application Module

The following steps show how to create a JSP project and application. This practice requires you to have created the BC4J project in the first hands-on practice in this chapter.

1. On the LocDept2WS.jws node, select New Empty Project from the right-click menu.
2. Enter "LocDeptJSP" for *Directory Name* and "LocDeptJSP.jpr" for *File Name* and click OK.
3. Click the LocDeptJSP.jpr node and select New from the right-click menu. Expand the Web Tier node and select JSP Page from the JavaServer Pages (JSP) category. Click OK.
4. In the New JSP dialog, leave the default setting for the *Directory Name* field. Enter "LocDept" in the *File Name* field and click OK. The Code Editor and Component Palette will open automatically and you should now see the LocDept.jsp file added to the project.

NOTE
The Component Palette for JSP files appears on the right side of the IDE by default.

5. In the Code Editor, change "Hello World" under <TITLE> to "Department Locations".
6. In between the <H2> tags, replace "the current time is:" with "Department Locations".
7. Delete everything between the </H2> and </BODY> tags, including the "P" tags. The Code Editor should now look like the following:



8. Position the cursor in the blank line before the </BODY> tag (add a blank line if there is none). Select BC4J Connections in the Component Palette pulldown.
9. Click ApplicationModule in the Component Palette. In the ApplicationModule dialog, if a data model definition appears, select LocDeptModule from the pulldown.

   Additional Information: If no data model definition is present, click New to start the BC4J Client Data Model Definition Wizard. Click Next on the Welcome page. Accept the defaults on the Definition page. Ensure that LocDeptModule appears in the *Definition Name* field. Click Next and Finish.
10. Click Next and Finish to complete the Application Module Wizard. The source code editor will now show the ApplicationModule tag.
11. Click Save All.

What Just Happened? You created the project and added a reference to the BC4J application module. This phase created a shell JSP and attached it to the same application module used for the Java application in the previous hands-on practice. The JSP is now ready to have UI components added to it that will connect to your BC4J application in the same way that the Java application did.

II. Add a Data Source and Data Tags

Just as you did in the BC4J and Java application projects, you will use the Locations and Departments information in the HR schema for your JSP project. Now that you have defined the BC4J application module, you can define which view objects you will use in this JSP. The data tag that points to a view object is called DataSource and you will add a data source for the LocationsView and another data source for the DepartmentsView.

You will also add data tags to display navigation buttons that are associated with a view object. To display the data, you will add a data tag to display a single Locations record and another data tag to display multiple Departments records. The functionality for retrieving the data and constructing the HTML display is built into the data tag code and the representation of that code in the JSP is relatively simple.

1. Make sure that the cursor in the Code Editor is on a blank line after the jbo:ApplicationModule line. Click DataSource on the BC4J Connections Component Palette page. Select LocationsView1 and click Next.
2. Enter "locationsData" in the *id* field. Leave the other values blank. Click Finish. The Code Editor will now show the DataSource tag.
3. Click DataSource on the BC4J Connections page of the Component Palette. Select DepartmentsView1 (the child under the LocationsView1 node) and click Next.
4. Enter "departmentsData" in the *id* field and "8" in the *rangesize* field. Leave the other values blank. Click Finish. The Code Editor will now show the DataSource tag.
5. Select the BC4J Component Tags page from the Component Palette pulldown. Click DataHandler in the Component Palette. Select "LocDeptModule" in the *appid* pulldown and click Finish. The DataHandler tag will be added to the code. This tag manages the updating of the display when you click the Next and Previous links.
6. Click Save All.
7. You can now add the display data tags to the JSP. Click DataNavigate in the Component Palette. Select "locationsData" in the *datasource* pulldown. Click Finish to add the tag to the code.
8. Click DataRecord. In the *datasource* field, select "locationsData" from the pulldown. Click Finish. Another tag will be entered into the JSP.
9. Click Save All.
10. Add a line under the last DataRecord tag and type "<h2>Departments</h2>."
11. In a blank line after the `</h2>` tag, click DataTable in the Component Palette.
12. Click in the *datasource* field and select "departmentsData." Leave the other default settings. Click Finish.
13. Click Save All.
14. Select LocDept.jsp in the Navigator and click the Run icon. If you see the Default Run Target dialog, select the LocDept.jsp file from the pulldown (this associates the file that will be run when you run the project) and click OK.

Additional Information: Your default browser will open, and an application such as the one in Figure 3-9 will appear. This may take some time.

15. Try the Next and Previous browse links in the Locations area to see how the navigation data tag works and how the data table will display department records appropriate to the location.
16. Close the browser window.

What Just Happened? In this phase, you added a number of UI components to support a simple master-detail JSP. If you compare this to the JSP pages generated in Chapter 1, you will see that this is only a small part of a full JSP application.

III. Modify the Display

In this phase, you will see the effects of using a style sheet to determine the look and feel of the JSP page. You will also see how data tags can be added or changed to alter the JSP page's behavior.

Apply a Style Sheet

You can modify the look and feel of your JSP by using a predefined style sheet. You can also create your own style sheets to give a more distinctive look to your applications. For this practice you will apply the same style sheet used by the JSP wizards.

1. Add the following line in the heading section after the <meta> tag:

```
<LINK REL=STYLESHEET TYPE="text/css" HREF="bc4j.css">
```

2. Select the LocDeptJSP.jpr project node and select **File | Import** to access the Import dialog. Select Existing Sources and click OK to start the Import Existing Sources Wizard.

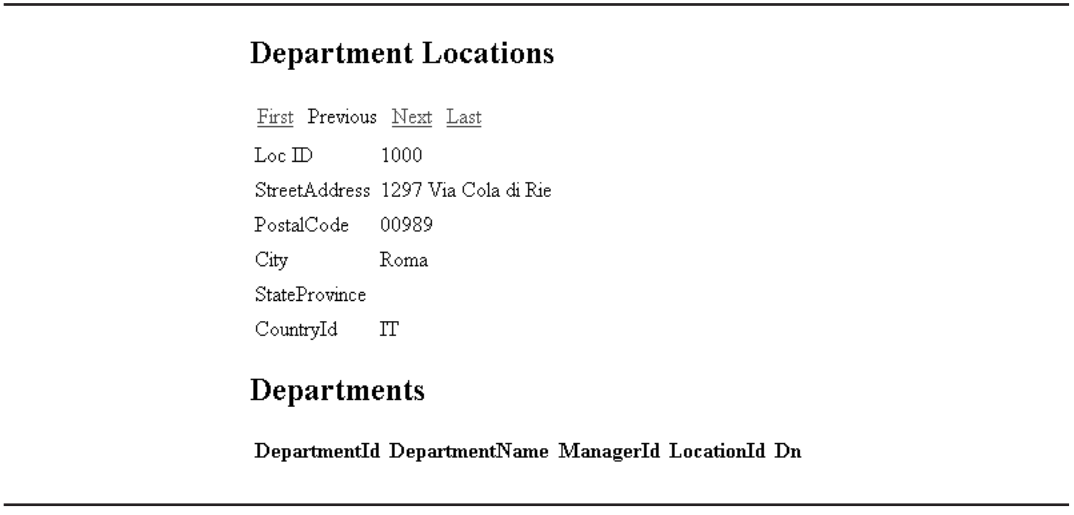


FIGURE 3-9. *Generated JSP*

3. Click Next if the Welcome page appears. Click Add and navigate to the JDEV_HOME\jdev*systemXX*\templates\common\misc directory where the "XX" in *systemXX* represents a number.
4. Select the bc4j.css file in this directory and click Open.
5. Check the *Copy Files to Project Directory* checkbox. Enter the directory path JDEV_HOME\jdev\mywork\LocDept2WS\LocDeptJSP\public_html.
6. Click Next and Finish to close the wizard. The file will be copied to the project.
7. Click Save All.
8. Select the LocDept.jsp node and click Run. Your default browser will open, and you will see something like Figure 3-10.

Modify the Label Properties

By default, the labels for the fields use the attribute names from the view object in the BC4J layer. These labels are not necessarily the most intuitive to the user. However, there is a way to modify the display value associated with the attribute so that the data tag will use a friendlier label. This technique requires modification of the BC4J layer because the data tag reads all information about the data elements from that layer. The benefit of this technique is that all applications that use the same BC4J project will be able to take advantage of the prompt property because it is part of the common layer.

1. Open the LocDeptBC4J.jpr node in the LocDept2WS.jws workspace and navigate to LocationsView under the locdept package node.
2. Double click the LocationId object icon in the Structure window to display the Attribute Wizard for LocationId.

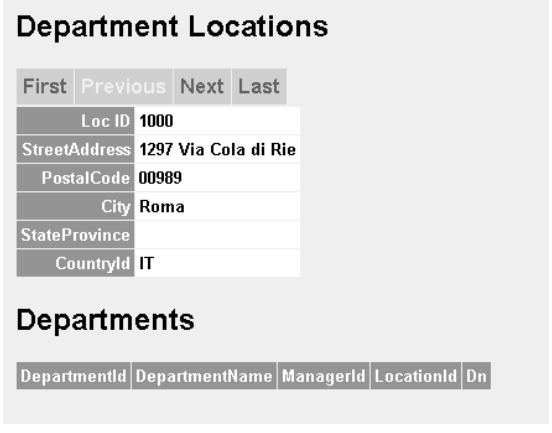
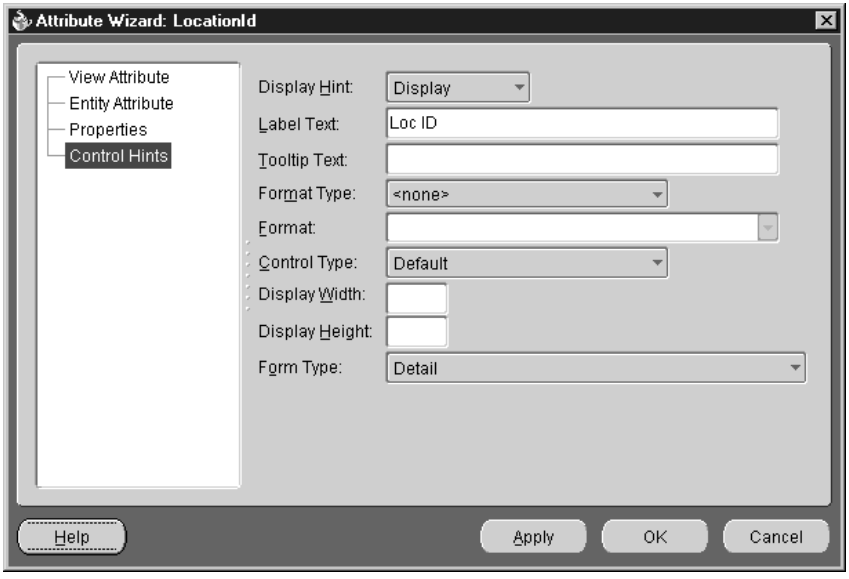


FIGURE 3-10. *JSP with style sheet applied*

3. Click the Control Hints node. Type "Loc ID" in the *Label Text* field as shown here:



4. This property holds the text that is used when the data tag displays the attribute. Click Apply and OK.
5. Click Save All.
6. Close the browser and click the Run Manager tab in the Navigator. If the Run Manager is not displayed, select **View | Run Manager**.
7. On the Embedded OC4J Server node, select Terminate from the right-click menu. This stops the OC4J server. You need to restart the server because the changes in the BC4J objects will not be read otherwise.
8. Click the System tab in the Navigator and select the LocDept.jsp node in the LocDept2WS.jws workspace. Click the Run icon to run the JSP page and verify the change.

You can modify the label properties of the other attributes in the same way. This does not require changing any code in the application, and any other JSP applications that use these same data tags will pick up the change automatically.

What Just Happened? You created an empty JSP application and added BC4J data tags for displaying a master-detail application showing a single location record and multiple department records. In addition, you altered the label of an attribute using BC4J properties.

When you use data tags, adding a data source to a JSP page is just a matter of being sure that your BC4J layer is set up. The database connection and table details are stored in the BC4J layer, which simplifies the code you have to write for the user interface. The data tag is attached to BC4J with the application module name. Chapters 21–24 explore JSP pages in more detail.