## Contents

Send Us Your Comments ...................................................................................................................... xi

Preface ................................................................................................................................................ xiii

### Part I Guidelines for Building Applications

#### 1 Managing Developer/2000 Applications

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 The Software Development Lifecycle: An Overview</td>
<td>1-20</td>
</tr>
<tr>
<td>1.1.1 The Developer/2000 management strategy</td>
<td>1-21</td>
</tr>
<tr>
<td>1.1.2 About Project Builder</td>
<td>1-21</td>
</tr>
<tr>
<td>1.1.2.1 Understanding Project Builder terminology</td>
<td>1-22</td>
</tr>
<tr>
<td>1.1.2.2 How Project Builder affects existing development roles</td>
<td>1-24</td>
</tr>
<tr>
<td>1.1.3 Exploring Project Builder benefits</td>
<td>1-25</td>
</tr>
<tr>
<td>1.1.3.1 Associating modules with an application</td>
<td>1-25</td>
</tr>
<tr>
<td>1.1.3.2 Automating actions based on file types</td>
<td>1-25</td>
</tr>
<tr>
<td>1.1.3.3 Creating dependencies between modules</td>
<td>1-25</td>
</tr>
<tr>
<td>1.1.3.4 Assigning default connection strings to modules</td>
<td>1-26</td>
</tr>
<tr>
<td>1.1.3.5 Designating which modules are to be included in the final install set</td>
<td>1-27</td>
</tr>
<tr>
<td>1.1.3.6 Sharing and porting project and subproject registry files</td>
<td>1-27</td>
</tr>
<tr>
<td>1.1.3.7 Accessing other Developer/2000 and third party tools</td>
<td>1-27</td>
</tr>
<tr>
<td>1.1.3.8 Using source control packages with Developer/2000</td>
<td>1-28</td>
</tr>
<tr>
<td>1.2 Managing Project Documents During Design and Development</td>
<td>1-28</td>
</tr>
<tr>
<td>1.2.1 Installing Project Builder</td>
<td>1-29</td>
</tr>
<tr>
<td>1.2.1.1 Installing the project and user registries</td>
<td>1-29</td>
</tr>
<tr>
<td>1.2.2 Creating a project</td>
<td>1-30</td>
</tr>
<tr>
<td>1.2.2.1 Creating a project: Project Administrator</td>
<td>1-30</td>
</tr>
<tr>
<td>1.2.2.2 Creating a project: Team members</td>
<td>1-34</td>
</tr>
<tr>
<td>1.2.3 Working with projects and project documents</td>
<td>1-36</td>
</tr>
<tr>
<td>1.2.3.1 Working with projects: Project Administrator</td>
<td>1-37</td>
</tr>
</tbody>
</table>
## 3 Designing Visually Effective Applications

3.1 Understanding the Process ................................................................. 3-159
3.1.1 What are the stages? ................................................................. 3-161
3.1.2 Defining user requirements ......................................................... 3-161
3.1.3 Planning the user interface ......................................................... 3-162
  3.1.3.1 Creating your standards ...................................................... 3-163
  3.1.3.2 Considering portability ....................................................... 3-165
  3.1.3.3 Creating a prototype ........................................................... 3-165
3.1.4 Building the user interface elements .......................................... 3-167
  3.1.5 Gathering user feedback .......................................................... 3-167
3.2 Creating an Effective Form ............................................................... 3-168
  3.2.1 Understanding forms ................................................................ 3-168
    3.2.1.1 What is a module? ............................................................. 3-168
    3.2.1.2 What are forms, blocks, items, regions, and frames? ....... 3-169
    3.2.1.3 What are windows and canvases? .................................... 3-170
  3.2.2 Guidelines for building forms .................................................. 3-172
    3.2.2.1 Using object libraries ....................................................... 3-173
    3.2.2.2 Understanding basic design principles ............................ 3-174
    3.2.2.3 Adding color .................................................................... 3-176
    3.2.2.4 Creating canvases ............................................................ 3-177
    3.2.2.5 Creating windows ............................................................. 3-179
    3.2.2.6 Creating regions .............................................................. 3-180
    3.2.2.7 Adding items to blocks .................................................... 3-181
    3.2.2.8 Designing messages ......................................................... 3-185
    3.2.2.9 Implementing online help ................................................ 3-187
    3.2.2.10 Building effective menus ............................................... 3-188
3.3 Creating an Effective Report .......................................................... 3-188
  3.3.1 Understanding Reports ............................................................ 3-189
  3.3.2 Using Templates in Report Builder ........................................... 3-190
  3.3.3 Understanding Layout Objects .................................................. 3-191
  3.3.4 Controlling Layout Objects in Report Builder ......................... 3-191
    3.3.4.1 Using anchors ................................................................. 3-192
    3.3.4.2 Using the Print Object On and Base Printing On properties 3-193
Send Us Your Comments

Developer/2000: Guidelines for Building Applications
Part No. A57752-1
Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this publication. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most about this manual?

If you find any errors or have any other suggestions for improvement, please indicate the chapter, section, and page number (if available), and email them to d2kdoc@us.oracle.com.
The guidelines in this book are intended to help you fully exploit some of Devel-
oper/2000’s most powerful features. Whether you’ve been using Developer/2000
for years or are brand new to the product, the concepts and suggestions provided
in this book should make it easier for you to complete such tasks as deploying an
existing Developer/2000 application on the Web, designing an effective graphical
user interface, or tracking and managing the disparate modules that make up a sin-
gle application.

How does this book fit in with Developer/2000 online help? While you can always
rely on the online help to explain how to accomplish a given task or which options
are available to you within a product, this book helps you determine why you’d
want to choose one option over another and to understand the consequences of
each decision. Use this book to help develop your strategy; use the online help for
instructions on implementing that strategy.

These guidelines represent the combined experience of Developer/2000 customers,
marketing representatives, sales consultants, and the Oracle Applications group.
You may want to use these guidelines as the basis for developing your own com-
pany standards, or use them simply to augment the standards you already have in
place.

**Intended Audience**

This book is intended for anyone who uses Developer/2000 to build applications.
The needs of both novice and advanced users are addressed.
This book contains the following chapters:

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Description</th>
</tr>
</thead>
</table>
| Chapter 1, “Managing Developer/2000 Applications” | Explains how to use the tools currently available with Developer/2000 to set up and manage the development of applications. Topics include:  
- Setting up and administering projects  
- Enabling team development under a variety of networking scenarios  
- Source-controlling projects  
- Exporting projects between platforms  
Exporting projects to different environments during the application lifecycle |
| Chapter 2, “Deploying Applications on the Web” | Provides examples and guidelines for configuring, designing, and deploying applications on the Web. |
| Chapter 4, “Designing Multilingual Applications” | Explains how to design multilingual applications with Developer/2000. |
| Chapter 5, “Designing Portable Applications” | Discusses how to develop an application that can be easily ported across Windows 95, Macintosh, and UNIX. Also discusses developing for character mode terminals. |
Chapter 6, “Taking Advantage of Developer/2000’s Open Architecture”

Discusses how to use Developer/2000 to:

■ Create applications that include OLE objects and ActiveX controls.
■ Customize your applications with foreign functions.
■ Build and modify applications using the Open API.
■ Run applications against ODBC-compliant datasources.

Notational Conventions

The following conventions are used in this book:

<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>boldface text</td>
<td>Used for emphasis. Also used for button names, labels, and other user interface elements.</td>
</tr>
<tr>
<td>italicized text</td>
<td>Used to introduce new terms.</td>
</tr>
<tr>
<td>courier font</td>
<td>Used for path and file names.</td>
</tr>
<tr>
<td>COURIER CAPS</td>
<td>Used for:</td>
</tr>
<tr>
<td></td>
<td>■ File extensions (.PLL or .FMX)</td>
</tr>
<tr>
<td></td>
<td>■ Environment variables</td>
</tr>
<tr>
<td></td>
<td>■ SQL commands</td>
</tr>
<tr>
<td></td>
<td>■ Built-ins/package names</td>
</tr>
<tr>
<td></td>
<td>■ Executable names</td>
</tr>
</tbody>
</table>
Part I
Guidelines for Building Applications
Managing Developer/2000 Applications

One of the most important aspects of application development is managing the modules that make up an application. Large applications can consist of literally thousands of modules, and millions of lines of code. In addition, modules which are important to the project as a whole but which are not compiled into the application itself (such as design specifications, test scripts, and documentation) must also be tracked and maintained.

This chapter explains how to use Developer/2000 to help you manage the application development process.

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Section 1.1, “The Software Development Lifecycle: An Overview”</td>
<td>Briefly covers the major milestones of application development and discusses Developer/2000’s Project Builder within that framework.</td>
</tr>
<tr>
<td>Section 1.2, “Managing Project Documents During Design and Development”</td>
<td>Discusses how Developer/2000 can help manage documents during development of an application.</td>
</tr>
<tr>
<td>Section 1.3, “Managing Project Documents During the Test Phase”</td>
<td>Discusses how Developer/2000 can help ensure that your QA group tests the correct configuration of project documents during the test phase.</td>
</tr>
<tr>
<td>Section 1.4, “Managing Project Documents During the Release Phase”</td>
<td>Discusses how Developer/2000 can help ensure that an installable version of your application is delivered to your customers.</td>
</tr>
<tr>
<td>Section 1.5, “Deploying Completed Applications”</td>
<td>Discusses how to turn your own application into one that is installable by the Oracle Installer.</td>
</tr>
</tbody>
</table>
1.1 The Software Development Lifecycle: An Overview

Application development typically occurs in four phases:

- **Design.** The initial specification for the application is developed. This specification can be based on a variety of sources: customer feedback, input of project management or development team members, requests for enhancement, necessary bug fixes, or systems analysis.

- **Develop.** Individual modules are created or modified, possibly incorporating a wide variety of languages, tools, or platforms.

- **Test.** The modules are tested. This generally occurs in two stages: *unit test* and *system test*. Unit test is testing at a modular or functional level; for example, testing UI elements such as menus or buttons. System test tests the integration of major portions of the code; the backend with the UI, for example.

- **Deploy.** The modules are packaged together in an installable form and delivered to customers.

*Figure 1–1 The phases of the development lifecycle: input and deliverables*

As the application grows in size and complexity, the four phases are repeated iteratively, and the amount of information produced (actual code, bug reports, enhancement requests, etc.) grows. Yet all input and deliverables for all phases must be tracked and maintained to ensure the integrity of the final deliverable: the application your customer installs.
This chapter discusses how to use Developer/2000 to manage your application’s code base and maintain version integrity.

1.1.1 The Developer/2000 management strategy

In any development project, management tasks can be split roughly into two categories:

- Project management, which includes allocating the necessary equipment, budget, and person-hours of work necessary to complete the development of the application.
- Software configuration management, which includes assigning modules to developers, determining dependencies among modules, maintaining the code under development, and version control.

With Release 2.0, Developer/2000 introduces Project Builder, a powerful tool that enables you to simplify your software configuration management tasks so you and your team can focus on your primary objective: designing, coding, and testing applications.

1.1.2 About Project Builder

To help simplify your software management tasks, Project Builder provides the means for you to:

- Associate modules with an application or component of an application.
- Automate actions based on file types.
- Create dependencies between modules and indicate how changes cascade; in other words, show which modules need to be recompiled based on changes to other modules.
- Assign default connection strings to modules.
- Designate which modules are to be included in the final install set.
- Share projects and subprojects among team members and port them to different environments.
- Invoke other Developer/2000 and third party tools from the Project Builder user interface.

These features are described in detail in Section 1.1.3, “Exploring Project Builder benefits”. If you’re unfamiliar with Project Builder terminology, however, it’s a good idea to read through Section 1.1.2.1, “Understanding Project Builder terminol-
ogy” before proceeding. This section defines some basic terms which provide the context for a discussion of Project Builder’s features.

1.1.2.1 Understanding Project Builder terminology

Project Builder is based upon the concepts of projects and subprojects:

■ Projects are collections of pointers to the modules and files that are part of your application.

■ Subprojects are projects contained within other projects, providing a finer level of organizational granularity. Often the organization of files into subprojects mirrors the organization of files into subdirectories, but this is not a requirement.

In addition to projects and subprojects, these terms are also central to a solid understanding of Project Builder:

■ Types. A type is the basis of every entry, and controls the actions that are available in Project Builder. Project Builder types recognize their associated file types primarily by default extension; for example, `.TXT` for text files. Project Builder predefines types for many commonly used files, such as forms documents (`.FMB`), text files, and C source files.

■ Entries. The components that make up a project are known as entries. A entry is simply a description of a file that is part of a project. Each entry is fully described in the associated Property Palette, which lists the entry’s type, location in the file system, size, and when it was last modified. The actions and macros (see below) for the entry are also defined.

It is important to remember that an entry is not the file itself; rather, it is a description of the file. So when you delete an entry from a project, you are simply telling Project Builder that the file is no longer part of the project. The file itself is not deleted.

■ Actions. Actions are command strings that apply to files of a given type; for example, the Edit action for a text item may be the command string that invokes Notepad or WordPad.

■ Macros. Macros are variables you can use to modify actions. A macro may be either a constant or a simple expression (which, in turn, may contain other constants and/or expressions). For example, Project Builder inserts all the information you’ve specified for connecting to a database into the `ORACONNECT` macro, which is included in all commands that might require you to connect. The information in the macro is then inserted into the action so you can log on automatically.
Just as you might use environment variable in scripts or batch files to conveniently modify a script’s actions without editing the script itself, so you can use macros to customize actions without having to edit the action themselves. For example, you might define a macro to determine whether to compile your application in Debug mode or Optimized mode. In preparation for building the deployment version of the application, you would simply change one macro definition to switch off Debug, rather than having to find and modify every type whose Build command made use of the Debug flag.

- **Global registry.** The Global Registry contains the pre-defined Project Builder types.
- **User registry.** Each user has a user registry in which to define new types, redefine existing types, and modify or create actions or macros.
- **Project registry file.** The project registry file contains information necessary to track a project, including pointers to modules contained within the project, default connection strings, and a pointer to the “home” directory for the project.
- **Project items:** Project items are all the objects in a project, such as types, actions, macros, and modules.

The Project Builder interface provides three tools for manipulating the items that make up a project:

- The **Project Navigator** furnishes a familiar “navigator” or “explorer” style interface with which you can view the modules in your application. You can also launch editing tools directly from the Project Navigator.
- The **Property Palette** enables you to examine and modify the properties of selected items.
- The **Launcher**, a secondary toolbar, provides another means of accessing development tools. You can even add buttons to the Launcher and associate them with your favorite third-party tools.
The Software Development Lifecycle: An Overview

Figure 1–2 depicts all three of these tools.

![The Project Builder user interface](image)

**Figure 1–2** The Project Builder user interface

1.1.2.2 How Project Builder affects existing development roles

Certain roles must be filled to keep the application development effort going smoothly. Some, such as project manager, development manager, and team leader, are common roles within development groups and require no definition. However, with Project Builder one new role exists—that of **project administrator**.

A project administrator is charged with creating projects and making them available to developers. The project administrator maintains the Global Registry and modifies it as necessary, exporting the changes to the developers on the team. He or she may also export the project information to different environments, such as test environments, or other platforms for cross-platform development.

The work the project administrator does when managing projects may affect the roles of the following team members:

- Developers
- Source control administrator
- Testers (QA)
Of course, the precise duties of each team member vary from development group to development group. A team member may also take on more than one role; for example, a team leader may also be a project administrator, or a developer may be in charge of source control.

1.1.3 Exploring Project Builder benefits

Now that you are familiar with basic Project Builder terminology (if you’re not, see Section 1.1.2.1, “Understanding Project Builder terminology”), let’s examine the benefits Project Builder provides.

1.1.3.1 Associating modules with an application

You can associate all of the modules in an application with the application itself simply by adding the modules to the same project. This allows you to track a large application as a single entity, determine the dependencies between modules, and so on.

1.1.3.2 Automating actions based on file types

Project Builder ships with an extensive list of types, to which are assigned default actions (such as Open, Edit, or Print). When you select a module, then click the right-mouse button, a pop-up menu displays the actions associated with that type. By default, the actions included in a type definition apply to all modules of that type in a project. You can also modify and add to these actions.

Actions are simply command strings. One benefit to defining actions with the actual command strings (besides simplicity, of course) is that an action can be associated conceptually with several different types. For example, editing a Word document requires a different tool than editing a text document, yet conceptually the two edits are very similar. Project Builder can associate the same Edit command with two different types, but no confusion results because the Edit command has a different command string for each type. In this way, a single command executes an appropriate action no matter what type of module you’re working with.

1.1.3.3 Creating dependencies between modules

Knowing which modules depend on which other modules is necessary to determine when modules need to be recompiled as a result of a change. It’s also the key to managing the impact of changes; for example, if a library changes, which forms are now out-of-date?
Project Builder includes the dependencies for module types in their type definitions. Thus, it can recognize dependencies between existing modules in a project. Since it can also track modifications to modules, it automatically recompiles changed modules and the modules dependent on them.

In fact, Project Builder can recognize dependencies that do not yet exist within the project and create markers for them. These markers are called implied items. For example, suppose your project contains an .FMB file, defined by the Project Builder type “Form Builder document.” The “Form Builder executable,” or .FMX file, may not exist—you may not have generated it yet. But Project Builder knows the existence of this .FMX file is implied by the existence of the .FMB file, and creates an implied item to mark it.

To determine the existence of an implied item, Project Builder correlates the value of the property Deliverable Type for each defined type with the input items, or source, required for the Build From <type> action for each defined type. In our example above, the Deliverable Type property for the “Form Builder document” type is defined as “Form Builder executable,” or .FMX. The Build From <type> action defined for a Form Builder executable is “Build From FMB”. This means .FMB files are the input items for creating .FMX files, and, conversely, .FMX files are targets for .FMB source.

The chain of implied items can consist of multiple files. For example, suppose you add a C source file to a library file. In this case, Project Builder adds modules of whatever other types are necessary to get a complete path of Build From <type> actions from one file type to the other (like an object file).

While Project Builder detects dependencies only between compilable modules and their resultant executables, you can set dependencies manually by adding modules to a project below the item dependent on them. For example, if an .FMB is dependent on a PL/SQL library, you can add the .PLL to the project below the .FMB, and Project Builder will recognize the dependency.

To see the dependencies in your project, choose Navigator → Dependency View. This displays modules in the Project Navigator in the order in which they depend on each other, with project nodes at the highest point in the hierarchy, followed by target nodes, followed by buildable components of those targets (input items).

1.1.3.4 Assigning default connection strings to modules

With Project Builder, you can define all your most-used connection strings and store their definitions under the Connections node in your user registry. You can then assign a connection to a module by dragging the connection from the Connections node and dropping it on the module. When you need to edit that module—for
instance, a form—you can select the form in the Project Navigator and choose **Edit** from the pop-up menu. Project Builder automatically opens Form Builder and connects to your database for you.

1.1.3.5 Designating which modules are to be included in the final install set

Project Builder makes it easy to determine and track the modules that will be included in your final install package (for example, `.exe` files, `.dll` files, and `.hlp` files). To earmark a file for delivery, set the **Deliver File** property to Yes. When you’re ready to create an install package, the Deliver action packages all modules for which the **Deliver File** property is set to Yes into a single unit.

**Note:** You can set the **Deliver File** property for a type or for individual project items.

1.1.3.6 Sharing and porting project and subproject registry files

Project Builder enables you to export the information about a project to other team members and to other platforms. Information about types, actions, macros, and project registry files—including all the customizations you’ve made—can be written to a text-based export file which you can then import to other environments and other platforms. This enables cross-platform development and testing.

1.1.3.7 Accessing other Developer/2000 and third party tools

You can access other tools from the Project Builder user interface through several means:

- **Actions**, which you access by selecting a module in the Project Navigator and clicking the right mouse button. A pop-up menu is displayed, listing all the actions associated with the selected item; the actions listed invoke whatever tools are specified in the command strings. You can also double-click an entry in the Project Navigator to invoke its default action.

- The **Compile**, **Deliver**, and source control actions, which you access from the **File → Administration** menu. These actions launch whatever tools are associated with them.

- The Launcher toolbar, which launches many Developer/2000 components such as Form Builder, Report Builder, and Graphics Builder. You can also add your own buttons to the Launcher toolbar and associate them with your favorite third-party tools.
### 1.1.3.8 Using source control packages with Developer/2000

Developer/2000 provides interfaces to these source control packages:

- PVCS from Intersolv
- Clearcase from PureAtria
- Versions, the source control component of StarTeam, from StarBase

You can also use other source control tools by modifying the source control actions provided with Project Builder to point to them.

Since a variety of source control packages are available and can be used with Developer/2000, specific instructions for source-controlling your projects are beyond the scope of this chapter. However, general guidelines will be provided where applicable.

### 1.2 Managing Project Documents During Design and Development

Much has been written about the importance of design in the success of an application. Deliverables during the design phase can include design documents and specifications, meeting minutes, UI prototypes, results from customer surveys (if the application is new), user tests and lists of enhancement requests (if the application is to be revised)—all documents that can be added to and tracked within a project.

This means the project administrator for the development effort should be identified early in the design process and begin creating the project immediately. (See Section 1.1.2.2, “How Project Builder affects existing development roles” for information on the role of the project administrator.) This section describes the role of the project administrator and the members of the development team in setting up Project Builder to manage a project during the design and development phase. Specifically, this section addresses:

- “Installing Project Builder”
- “Creating a project”
- “Working with projects and project documents”
- “Managing projects and project documents across multiple platforms”

**Note:** The steps involved in accomplishing simple tasks with Project Builder are in the Project Builder online help and will not be included in this chapter.
1.2.1 Installing Project Builder

Project Builder is automatically installed to `ORACLE_HOME\PJ10`, where `PJ10` is a variable that determines the search path for the various Project Builder registry files.

Noteworthy files found in this directory are:

- Project Builder executable (`PJ10.EXE`)
- Global Registry file (`TYPESnn.UPD`), where `nn` indicates the national language
- Default user registry files (`PJUSERnn.UPD`), where `nn` indicates the national language

Perhaps the most important issue to address during Project Builder installation is how you want to make these various files available to team members. Section 1.2.1.1, “Installing the project and user registries” discusses your options.

1.2.1.1 Installing the project and user registries

Project Builder depends on native file sharing protocols for its security. This can make project files vulnerable to accidental changes, which is something to keep in mind as you decide how to configure the Global Registry and user registries. Table 1–1 lists the available options.

<table>
<thead>
<tr>
<th>Option</th>
<th>Pros</th>
<th>Cons</th>
<th>Recommendation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Install Project Builder with the Global Registry on a shared network drive and the user registries on local machines.</td>
<td>If your team is networked, developers can access a single copy of the Global Registry. This ensures that all versions of the Global Registry in use are up-to-date.</td>
<td>If all team members have write access to the Global Registry, it can be accidentally overwritten.</td>
<td>To prevent the Global Registry from being accidentally overwritten, install it in a directory to which only you have write access.</td>
</tr>
</tbody>
</table>

Table 1–1 Registry installation options
1.2.2 Creating a project

The following sections focus on the creation of a single project for distribution to a team of developers. However, this may not be the best option for your group. If the application being developed is very large or if components are to be split up among team members, you may choose to create several separate, smaller projects, the contents of each determined by the responsibilities of each developer or group of developers.

If you decide to distribute a single project, note that Project Builder projects will accept pointers to modules that do not exist in the specified location. (You can determine whether a module exists by examining its information in the property palette; **Time created/modified** and **File size (bytes)** will be blank). This means you can distribute a single large project without requiring all team members to have all modules available.

Creating a project is an ongoing task that requires the participation of both the project administrator as well as each member of the development team. This section describes the responsibilities unique to each role.

### 1.2.2.1 Creating a project: Project Administrator

As project administrator, your role goes beyond creating a project registry file and deciding what to include in the project. Whether you use the Project Wizard pro-

<table>
<thead>
<tr>
<th>Option</th>
<th>Pros</th>
<th>Cons</th>
<th>Recommendation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Install Project Builder with copies of the Global Registry available to each team member, in addition to their own user registries.</td>
<td>You can propagate updates to the Global Registry simply by making a copy of the changed file available to your team members (if they are on the same platform).</td>
<td>The individual Global Registries are not safe from accidental overwrites or deletions.</td>
<td>Use Project Builder’s Export facility to propagate changed registry files instead of providing copies. A more rigorous process may help discourage a casual attitude toward registry files.</td>
</tr>
<tr>
<td>Install Project Builder with the Global Registry and a single user registry shared among team members.</td>
<td>The types, actions, projects, and project modules are at risk for conflicting modifications.</td>
<td>Don’t choose this option. But if you must, have members of your development team edit only the modules, not the project itself.</td>
<td></td>
</tr>
</tbody>
</table>
vided by Project Builder to create the project, or create a project registry file and manually edit the various properties, prior planning is highly recommended before you complete the following tasks:

1. Create the project:
   a. Set up the project’s directory structure.
   b. Add modules.
   c. Establish default actions, macros, and connection strings.
   d. Set necessary dependencies manually.

2. Work with the source control administrator to set up a concurrent source control project.
   a. Define new types and edit existing ones.
   b. Customize actions and macros.
   c. Create reusable connections.

3. Make the project available to team members.

The next few topics provide recommendations for completing each of these tasks.

1.2.2.1.1 Step 1: Creating the project

The Project Wizard provides an easy-to-use interface for creating a project. You can also create a new project without the Project Wizard (using the New Project tool on the toolbar) and set project properties in the Property Palette.

At its simplest, a new project is a default project registry file primed with information about the Global Registry, but little else. Project Builder needs a bit more information before it can keep track of your project, as discussed in the next few topics.

Step 1a: Set up the project’s directory structure

The directory structure of a project can have far-reaching consequences. For example, suppose a project contains modules that are located in a directory that’s not a child of the project directory. Now suppose you create actions that search for and modify project modules. How will you find the “orphan” modules? Create alternate actions with hardcoded paths? Not portable. Search from the root? Not efficient.

Recommendations:
- Place modules in the project directory or in a directory that’s a child of the project directory (a good choice when adding subprojects).
As much as possible, organize your projects and subprojects so that they mirror your actual directory structure.

The standard method of adding modules to a project is the Add Files dialog. Note that the dialog always inserts the full path unless the module you want to add is in the project directory.

In all other instances, the dialog forces a full path. In that case, you can change it to a relative path by selecting the item (for example, a subproject) in the Project Navigator, displaying the Property Palette, and manually editing the Filename property.

**Step 1b: Add modules**

Once you have planned the directory structure, you can add modules to the project.

**Recommendation:** Use subprojects whenever possible to help organize your project. But don’t simply group together all forms or all reports. Group the modules into components; for example, you might create a subproject for all the modules in a large form, including .FMB files, .FMX files, PL/SQL libraries, menus, bitmaps, icons, etc. This enables you to more easily create some necessary dependencies not detected by Project Builder.

**Step 1c: Establish default actions, macros, and connection strings**

This step involves making site-specific edits to actions and macros; for example, changing build actions to use the compilers and compiler options that are standard at your site. If you have not already done so, you can also create connection strings for commonly used databases containing test data or necessary tables.

**Step 1d: Set necessary dependencies manually**

Project Builder can recognize some dependencies between modules (it knows that .FMX files are built from .FMB files, which are built from .FMT files), but only the dependencies it can deduce by cross-referencing the Deliverable Type and the Build From <type> actions.
Other dependencies may exist as well: dependencies on PL/SQL libraries, menus, icons, and so on. You can tell Project Builder about these dependencies by creating entries for the modules on which a module is dependent below the entry for the dependent module, as shown in Figure 1–3, “Manually added dependencies”.

![Figure 1–3 Manually added dependencies](image)

This figure illustrates NAVWIZ.FMB’s dependency upon WIZARD.PLL, NAVIGATE.PLL, and NAVWIZ.PLL.

To establish a manual dependency, select the entry for the dependent module in the Project Navigator and choose Add Files to Project. Entries for the modules you select are added to the Project Navigator under the dependent module, and Project Builder is notified of the dependency. (To see a graphical representation of the dependency, make sure Navigator → Dependency View is selected.)

1.2.2.1.2 Step 2: Work with the source control administrator

After you create your project, you’re ready to introduce a source control package. Many third-party source control packages also implement the concept of projects.

**Recommendation:** Work with your source control administrator to set up a source control project that mirrors your development project in Project Builder.

When setting up a project to source control a complex application, remember to include the non-obvious modules as well. For example, when checking in a form, don’t forget menus, PL/SQL libraries, user exits, icons, or special fonts you use. Applications running on Windows may use OCX or ActiveX controls that should be source-controlled as well.
1.2.2.1.3 Step 3: Make the project available to team members

Once you’ve done the preliminary work of creating the project and establishing source control, it’s a good idea to export all project information to a project export file and notify team members of its location. They can then import the project.

It is possible to notify team members of the location of the actual project registry file, but remember that Project Builder uses your operating system’s own security features to protect your project modules from being deleted or overwritten. Simple deletes and overwrites are possible. To maintain the integrity of your projects, follow Project Builder’s own process for updating projects, and always import and export modifications to the project instead of simply distributing changed registry files.

When you notify your team members of the location of the project export file, you should also notify them of the directory structure you’ve set up so they can mirror that structure on their development machines. The easiest course for setting up the project is to have all team members map the project location to the same project directory on their machines, since mappings to different project locations would require separate copies of the project registry file with different values for the Project Location: Q:\myproj, R:\, etc.

Team members can then check out the modules they have been assigned.

1.2.2.2 Creating a project: Team members

After the project administrator has completed the tasks described in Section 1.2.2.1, “Creating a project: Project Administrator”, project team members can fine-tune the work. If you are a project team member, you can expect to:

1. Set up your directory structure and import the project
2. Customize your user registry
   a. Define new types and edit existing ones
   b. Customize actions and macros
   c. Create re-usable connections
3. Check out your assigned modules

1.2.2.2.1 Step 1: Set up your directory structure and import the project

When your project administrator informs you that the project is available, it’s time to import the project information and set up your working directories with the modules you’ve been assigned.
Recommendation: File management is easier if you set up your directory structure to mirror what your project administrator has already created for the project.

1.2.2.2 Step 2: Customize your user registry

One of the first things to do when setting up a project is to customize your user registry.

Step 2a: Define new types and edit existing ones

If you want to add modules to your project that are of a type not represented in the Global Registry, you can define your own type in your user registry and assign actions, macros, and so on to it.

In addition, you may want to override a default command or macro for a particular type in the Global Registry. An easy way to accomplish this is to copy the type in the Global Registry, paste it into your user registry, and edit it. Now, all modules of that type in your project will inherit the modifications from the type in the user registry.

Recommendation: Notify your project administrator when you modify a global type by copying it into your user registry and editing it. Such a modification might be useful to the whole team.

Step 2b: Customize actions and macros

While you can customize the actions and macros associated with the types you add to your user registry, it’s important to remember that you can modify actions and macros at other points in the Project Builder hierarchy as well. Where you edit the item depends on the extent of the influence you want your change to have.

The following table lists all the locations you might find an action or macro, the scope of that action or macro, and what can override it.

<table>
<thead>
<tr>
<th>An action or macro assigned to:</th>
<th>Affects:</th>
<th>Unless overridden by:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global Registry</td>
<td>All items of type(s) to which it is assigned in all user registries and projects beneath the Global Registry.</td>
<td>Actions or macros in a user registry, project, subproject, or entry.</td>
</tr>
<tr>
<td>User registry</td>
<td>All items of type(s) to which it is assigned in all projects beneath the user registry.</td>
<td>Actions or macros in a project, subproject, or entry.</td>
</tr>
</tbody>
</table>

Table 1–2 Action and macro inheritance rules
Managing Project Documents During Design and Development

Step 2c: Create reusable connections

If you have your own set of tables with data you’ve created for testing purposes, you can add your own connections to the list provided by the project administrator. Once you’ve created the connections, you can assign a connection to a module by selecting the connection’s entry in the Project Navigator, dragging it to the project file entries, and dropping it on the entry for the module you’ve chosen. Now, when you select an action that opens a tool requiring a database connection, Project Builder logs on for you.

1.2.2.3 Step 3: Check out your assigned modules

Once you have your directory structure in place and the project imported, you can populate your workspace with the modules you’ve been assigned. The source control commands Check In, Check Out, and Source Control Options, accessible from the main menu under File → Administration, are associated with actions defined for each type. This means you can modify the actions, if necessary, to affect the results of the commands—though this is not recommended for source control.

1.2.3 Working with projects and project documents

When the project enters the development phase, maintaining the integrity of the project becomes increasingly important.

**Recommendation:** Only the project administrator should make changes to the project that affects multiple team members (such as modifying the Global Registry or adding new subprojects containing new modules).
1.2.3.1 Working with projects: Project Administrator

While the application is in development, as project administrator your role is to maintain and support the project. In addition, you might be in charge of managing development deliverables, or working with a development manager to do so. You might need to:

- Add new modules and dependencies
- Export modifications to the project registry file
- Apply version labels

1.2.3.1.1 Adding new modules and dependencies

Sometimes new modules must be added to a project after its initial creation, and dependencies added manually. The process for doing so is the same as when creating the initial project. For more information, see Section 1.2.2.1.1, “Step 1: Creating the project”.

1.2.3.1.2 Exporting modifications to the project registry file

Once you’ve added the new modules and made the necessary changes, you can export the changes and make them available to members of your team. The process for doing so is the same as when exporting the initial project. For more information, see Section 1.2.2.1.1, “Step 1: Creating the project”.

1.2.3.1.3 Applying version labels

Although you can try to keep various revisions synchronized with each other (for example, through a nightly check-in), often development on one module will be completed while another needs bugs fixed or headers changed. Synchronous revisions are generally impractical.

A better method is to synchronize versions by applying a symbolic version label to the group of revisions that mark the achievement of a significant milestone. Most major source control tools enable you to apply a symbolic label to a source control project.
1.2.3.2 Working with project documents: Team members

When your project is set up and your modules have been assigned, you can use Project Builder to:

- Edit modules
- Add modules and dependencies manually
- Build your project
- Check modules in and out

1.2.3.2.1 Editing modules

**Recommendation:** The most efficient way to use Project Builder to edit modules is to customize the actions associated with the types of modules you’ll be editing so they invoke the tools you want to use with the options you need. In addition, be sure to associate a connection string with either the individual module or the project. Then you can drag the connection from its location in your user registry and drop it on the module or project entry. Once your modules are prepared in this fashion, choosing a pop-up menu item or double-clicking on a project entry opens your module in the correct application. If necessary, you’ll already be logged on.

You can also use the Launcher to access development tools. The Launcher is shipped with toolbar buttons already set for the Developer/2000 tools, but you can add a third-party tool by creating a button and associating it with an executable.

**Note:** If you invoke a tool via the Launcher and then open a module, the tool will not be aware of any associated connection strings. You will need to log on to the database manually.

1.2.3.2.2 Adding modules and dependencies manually

See Section 1.2.2.1.1, “Step 1: Creating the project”, or contact your project administrator.

1.2.3.2.3 Building your project

The **Compile** commands—**Compile Selection**, **Compile Incremental**, and **Compile All**—are available from the main menu (under **Project**). They are also associated with an action—in this case, the **Build From <type>** action.

This means you can select a single command for any different module type and the module will be compiled according to the definition of the **Build From <type>** action—not for that particular type, but for the target you actually want to build.
For example, the Build From <type> action for an .FMX file invokes the Form Generator and creates the .FMX file from the corresponding .FMB. What the Compile command compiles is the .FMB, but how it compiles the .FMB is determined by the action associated with the .FMX that results.

You can modify the results of the Compile commands by modifying the definition of the Build From <type> action for the corresponding target.

Choose Compile Selection to compile selected modules, or force a compile of all compilable modules by choosing Compile All. Because Project Builder can detect when modules are out-of-date and need to be recompiled, you can compile only out-of-date modules by selecting the entry for the project containing them, then choosing Compile Incremental.

Note: The Compile commands are also available from the pop-up menu.

1.2.3.2.4 Checking modules in and out

If modules need conversion for source control (for instance, the source control only works on text and your modules are binary), you can edit the Check file into RCS action to automate the conversion to text before check-in.

You can also edit the Check file out of RCS action in a similar fashion to convert the text-based source controlled version of the module back to binary.

1.2.4 Managing projects and project documents across multiple platforms

Many applications today run on multiple platforms, with development taking place on a variety of platforms as well. Chapter 5, “Designing Portable Applications” can help you ensure that the application underlying your project is portable.

To ensure that your project is portable, too, Project Builder supports development on several major platforms. To do so, it must ship with a Global Registry that reflects the platform; in other words, the types defined must be found on that platform, and the actions and macros must be written according to the syntax rules of that platform. This means the Global Registry, and, by extension, all user registries and project registry files, are not portable.

However, you can export information about a project to a text file and import the text file to another platform, as discussed in Section 1.1.3.6, “Sharing and porting project and subproject registry files”.

1.2.4.1 Managing projects across multiple platforms: Project Administrator
If you are the administrator of a project undergoing development on multiple platforms, you can expect to:
■ Branch off a source control project to contain the code for the platform
■ Export projects and project information to alternate platforms

1.2.4.1.1 Branching off a source control project to contain the code for the platform
Work with your source control administrator to create a branching source control project that enables your team members to isolate the code for the new platform.

1.2.4.1.2 Exporting projects and project information to alternate platforms
Creating an export file for the purpose of distributing a project to another platform is no different from creating an export file to distribute to team members on the same platform. The export file created by Project Builder is a text file, easily transferred to the alternate platform.

1.2.4.2 Managing project documents across multiple platforms: Team members
The role of a team member working on development on an alternate or secondary platform is actually quite similar to the role of a team member developing on the base platform. However, there is one major difference: when you receive a project already created on a different platform, you can expect to:
■ Revise customized actions and macros to conform to platform requirements

1.2.4.2.1 Revising customized actions and macros to conform to platform requirements
Equivalent versions of pre-defined actions and macros, where they exist, are provided by Project Builder for all supported platforms. However, if some actions have been customized or new actions created, you will either need to edit the actions to make them work on the new platform or create equivalent new actions.

1.3 Managing Project Documents During the Test Phase
Though the test phase is often thought of as separate and distinct from the development effort—first you develop, then you test—testing is a concurrent process that provides valuable information for the development team.
There are at least three options for integrating Project Builder into the test phase:

- Your testers do not install Project Builder. You use Project Builder functionality to compile and source-control the modules to be tested and hand them off to the testers, whose process remains unchanged.

- The testers import the same project or projects that the developers use.

- You create a project based on the development project but customized for the testers (for example, it does not include support documents, specs, or source), who import it.

**Recommendation:** A combination of the second and third options works best. Associating your application with a project can be useful during the testing phase, as well. You can create actions to automatically run test scripts or add script types and make them dependent on the modules they are to test.

During unit test, testers can use the same project or projects as the developers, if the project is organized by functional units, or separate projects have been created for functional units. The project or projects can also be exported, so unit test can take place in a variety of environments and on a variety of platforms.

System test might require a new, stripped-down version of the development projects that includes only the modules being tested, especially if you need to concatenate several smaller projects.

### 1.3.1 On the development side

The goal of the development group in this phase of the process is to provide the test group with the modules to be tested in as smooth a manner as possible.

#### 1.3.1.1 The test phase: Project Administrator

The tasks involved in creating and exporting a project for testing purposes are the same as the tasks required when creating and exporting a project to a development team:

- Create a test project based on deliverable modules (optional)

- Create the test version

- Export the project to different test environments

### 1.3.2 On the test side

Although members of the test team generally are not responsible for any modifications to the modules of an application, they do have input (modules to test) and
deliverables (fully-tested modules and lists of bugs uncovered during the testing phase).

Project Builder can help the test team keep track of its input and deliverables in the same way it helps development team members. Testers can add scripts and logs to a project, modify actions to include debugging options, and add subprojects containing testing information.

1.3.2.1 The test phase: Testers
If you have decided to use Project Builder to help test your application, you’ll need to do some preparatory work that is very similar to that of the developers when they are first setting up their projects. You may need to:

- Import the test project and set up the testing environment
- Add test scripts and test data to the project
- Modify actions and macros to facilitate testing

1.3.2.1.1 Importing the test project and setting up the testing environment
The process of importing a test project and setting up a testing environment is the same as the process for importing a project and setting up the environment for development. See Section 1.2.2, “Creating a project”, for more information.

1.3.2.1.2 Adding test scripts and test data to the project
You may need to add some items, such as test scripts, to the project. In addition, you may need to add connection strings to database accounts containing test data.

Remember that you can automate the running of test scripts just as you can automate actions associated with the modules in your application.

1.3.2.1.3 Modifying actions and macros to facilitate testing
If actions specifying “run with debugging” have not already been provided, you can either modify existing actions to include a debug flag, or create new actions.

1.4 Managing Project Documents During the Release Phase
When your application has been thoroughly tested and is ready to release, Project Builder can help you simplify the process of delivering the application to customers.
1.4.1 On the development side

During the release phase, development hands off tested and verified versions of all modules necessary for installation to the releaser. Because Project Builder marks all modules to be included in the final application and associates special commands with them, this hand-off can be automated in the same fashion as other processes, such as compiling your project and source controlling it.

1.4.1.1 The release phase: Project Administrator

Once your project has been thoroughly tested and is ready for release, you have one remaining task: package the project.

1.4.1.1.1 Packaging the project

Project Builder packages an application by compressing all selected files into a .ZIP file on Windows, and a .TAR file on UNIX. To package an application for delivery, use the Deliver command, accessible from the main menu under File → Administration. Like the compile commands and the source control commands, the Deliver command is associated with an action: Deliver the selected file(s). The modules actually compressed by the Deliver action are determined by the value of the Deliver file property associated with each item (Yes to include the module in the package, No to leave it out).

Before packaging your files, define the PACKFILE macro to be the name of the .ZIP or .TAR file that is the result of the Deliver action.

As with all other actions, the Deliver action is completely customizable.

1.5 Deploying Completed Applications

After you have packaged your application, you’re ready to make it available to your customers. In addition to installing your application, your customers will also need to use the Oracle Installer to install the Runtime environment on which your application depends. To simplify the installation process for your customers, Developer/2000 provides the Oracle File Packager, with which you can make your own application installable with the Oracle Installer on Windows NT and Windows 95. When you’ve completed the steps in this section, your customers can install everything they need—your application, plus the required Runtime environment(s)—using a single mechanism.
1.5.1 Before You Begin

Before discussing how to package your application, it’s a good idea to familiarize yourself with the terminology and background information relevant to the installation/packaging process:

- Section 1.5.1.1, “Terminology”
- Section 1.5.1.2, “The Oracle Installer files”
- Section 1.5.1.3, “The contents of the DEPLOY directory”

1.5.1.1 Terminology

This table defines some important terms for the installation/packaging process:

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stage (or staging) area</td>
<td>The area on your PC or network where files and installation scripts are prepared, then copied to the distribution media.</td>
</tr>
<tr>
<td>Distribution media</td>
<td>The set of media (for example, tapes, CDs, or diskettes) from which users install your application.</td>
</tr>
<tr>
<td>Installable component</td>
<td>Any product (for example, Forms Runtime, GUI Common Files, and so on) that has its own set of Oracle Installer files (MAP, VRF, INS, and DEI).</td>
</tr>
<tr>
<td>Product file (PRD file)</td>
<td>A file that lists all installable components in a given staging area.</td>
</tr>
<tr>
<td>Oracle File Packager</td>
<td>A wizard that creates the product file and all the Oracle Installer files (MAP, VRF, INS, DEI) needed to make your Windows application installable through the Oracle Installer.</td>
</tr>
</tbody>
</table>
1.5.1.2 The Oracle Installer files

The Oracle Installer files control how and where an application is installed (and de-installed) on a user’s machine. While the Oracle File Packager creates the Oracle Installer files for you, you may have to make some slight modifications manually. If you just want to look at some sample installer files, take a look at:

/DEPLOY/RELEASE/FORWIN95/YOURAPP

/FORMSAPP
    FORMSAPP.MAP
    FORMSAPP.VRF
    FORMSAPP.INS
    FORMSAPP.DEI

/DEV2KAPP
    DEV2KAPP.MAP
    DEV2KAPP.VRF
    DEV2KAPP.INS
    DEV2KAPP.DE

All of these files are text files and should be viewable and editable in a text editor.

1.5.1.2.1 The PRD file

The PRD file lists all the installable components in a given staging area. It also identifies the base filename and location for the Oracle Installer files of each component. In other words, the PRD lists all the files that appear in the Available Products pane of the Oracle Installer. Its name reflects the platform it describes; e.g., WIN95.PRD and NT.PRD. There is one PRD file per staging area, per platform.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product number</td>
<td>Product number. You shouldn’t have to modify this.</td>
</tr>
<tr>
<td>Product</td>
<td>A unique name used to identify your application.</td>
</tr>
<tr>
<td>Parent</td>
<td>Leave as “root”.</td>
</tr>
<tr>
<td>Filename</td>
<td>Base filename of your MAP, VRF, INS, and DEI installation scripts.</td>
</tr>
<tr>
<td>Version</td>
<td>Version number of your application.</td>
</tr>
</tbody>
</table>

Table 1–3 Columns in the PRD file
1.5.1.2.2 The MAP file

The MAP file is a table that lists all the files that make up your application.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Label</td>
<td>Name of your application as it appears in the Available Products window of the Oracle Installer.</td>
</tr>
<tr>
<td>Location</td>
<td>Relative path to the directory that contains the installation script files (MAP, INS, VRF, and DEI) and all the files that make up your application.</td>
</tr>
<tr>
<td>Size</td>
<td>Total size of the installable component. Set automatically by the CHECKMAP utility.</td>
</tr>
<tr>
<td>Visible?</td>
<td>Makes the component visible (or not) in the Available Products window of the Oracle Installer.</td>
</tr>
<tr>
<td>Selected?</td>
<td>Makes the component selected (or not) in the Available Products window of the Oracle Installer.</td>
</tr>
<tr>
<td>Open?</td>
<td>Used for parent/child components. You should not need to modify this field.</td>
</tr>
<tr>
<td>Description</td>
<td>Describes your application.</td>
</tr>
<tr>
<td>Volume</td>
<td>Should match what appears in the Filename field. Not used for CD or LAN installations.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source</td>
<td>File to be copied to the user’s machine.</td>
</tr>
<tr>
<td>Destination</td>
<td>Directory to which the file is copied.</td>
</tr>
<tr>
<td>Group</td>
<td>Program group that will hold the program item(s).</td>
</tr>
<tr>
<td>Item</td>
<td>Name of the item or icon as it appears in the menu.</td>
</tr>
<tr>
<td>Command</td>
<td>Command that is executed when the item or icon is invoked. Appears in the format: command line working_directory alternate_icon Working_directory and alternate_icon are optional, however, if “command line” appears alone, it must end with a semicolon.</td>
</tr>
</tbody>
</table>

Table 1–4  Columns in the MAP file
Note: The Group, Item, and Command are required only for applications that appear in the Start menu. To see an example of how these fields are filled in, use your OS search capabilities to locate DEVDEM20.MAP, the map file for the Developer/2000 demos. (If you can’t find it, you may have to install “Developer/2000 Demos and Add-ons” from your Developer/2000 distribution media.)

1.5.1.2.3 The VRF file

The VRF file VeRiFies that all the correct dependencies are identified and installed. For example, by specifying that your application depends on Forms Runtime, your application’s installation process will automatically detect whether Forms Runtime is already present on a user’s machine. If it is not, Forms Runtime will be installed.

The VRF file also prompts the user for information, such as where the product should be installed. In addition, the VRF file sets up the user’s environment, defining such things as environment variables in the Windows registry.

1.5.1.2.4 The INS file

The INS file INStalls the files that make up an installable component, sets any needed environment variables, and registers the product with the Oracle Installer. It works in coordination with the MAP file and the VRF file.

1.5.1.2.5 The DEI file

The DEI file DEInstalls the files that make up an installable component. It also removes environment variables and unregisters the component after successful deinstallation. It works in coordination with the MAP file.

1.5.1.3 The contents of the DEPLOY directory

The DEPLOY directory provides everything you need to set up and customize your own staging area. Available wherever Developer/2000 was installed, the DEPLOY directory contains:

- The OISFP10 subdirectory, from which you can install the Oracle File Packager.
- The RELEASE subdirectory, which serves as a starting point for creating your own staging area.
- RELEASE/FORWIN95/WIN95.PRD, a PRD file that lists the installable components for Forms, Reports, and Graphics Runtime environments on Windows 95:
  - Required Support Files
Deploying Completed Applications

- System Support Files
- GUI Common Files
- Tools Utilities
- Forms Runtime
- Reports Runtime
- Graphics Runtime

- RELEASE/FORWINNT/NT.PRD, a PRD file that lists the installable components for Forms, Reports, and Graphics Runtime environments on Windows NT (see the previous bullet for a components list).

1.5.2 Making your application an installable product

This section contains instructions for creating a one-step or a multi-step installation process for your customers:

- One-step process: Your customers install your application and the Runtime environment they need from a single PRD file. Another way to think of this is that your customers install everything they need—your application, plus the required Runtime environment(s)—from a single invocation of the Oracle Installer.

- Multi-step process: Your customers install applications from many different staging areas, each of which has its own PRD file. This approach works well if you need to distribute many Developer/2000 applications, or if the required Runtime environment is already available to your customers from a common area.

Whichever process you choose, to make your application installable with the Oracle Installer, you will:

- Copy the DEPLOY/RELEASE directory to your machine to serve as a starting point for your own staging area.

- Use the Oracle File Packager to create the PRD, MAP, VRF, INS, and DEI files you need to make your application installable through the Oracle Installer.

- Copy your files from your development area to the staging area. From there you can copy the files to your distribution media.

The rest of this chapter contains specific instructions for completing these tasks.
1.5.2.1 Deploying your application on Windows

If your application is installable on Windows 95 and NT, you can use the Oracle File Packager to create the Oracle Installer files and to copy your files from your development area to the staging area. The following steps address both one-step and multi-step installations.

Step 1: Install the Oracle File Packager

1. From DEPLOY/OISFP10 (on your Developer/2000 distribution media), click SETUP.EXE to invoke the Oracle Installer. SETUP.EXE detects which operating system is running and launches the appropriate Oracle Installer.

2. Select Oracle File Packager from the list of installable products.

3. Complete the installation process as prompted.

Step 2: Prepare your staging area

1. Copy DEPLOY/RELEASE to a drive on your PC or a networked drive.

2. Create a subdirectory for your application under DEPLOY/RELEASE/FORWIN95, even if your application is targeted for the NT environment.

If you are staging more than one application, create a subdirectory for each.

Step 3: Move your files to the staging area and create the Oracle Installer files.

Repeat this step for each staging area you established in Step 2.

1. From the Start menu, select Oracle for NT or Windows 95, then select Oracle File Packager.

2. Follow the steps presented in the Oracle File Packager, using the online help to assist you.

Notes:

- The internal string you specify in Step 3 is prepended to your Oracle Installer files (MAP, INS, VRF, and DEI).

- When prompted for the Staging Area Location, specify the subdirectory under DEPLOY/RELEASE/FORWIN95.

Step 4: Merge your PRD file with NT.PRD and WIN95.PRD

This step creates a one-step installation process. If you’re creating a multi-step installation, go to Step 5.
1. Copy the line from your own application’s PRD file and paste it into RELEASE/FORWIN95/WIN95.PRD and/or RELEASE/FORWINNT/NT.PRD.

Step 5: Modify the Oracle Installer files

1. If you want your application to appear as an icon in the Start menu, add the Group, Item, and Command fields to the MAP file(s) for your application(s). To see an example of how to fill in these fields, use your operating system’s search capabilities to find the Developer/2000 Demos MAP file, DEVDEM20.MAP.

2. If you wish to establish some dependencies for your application, add them to the VRF file.
   
   For example, if you establish Forms Runtime as a dependency for your application, the installation process will automatically detect whether Forms Runtime is already present on a user’s machine. If it is not, Forms Runtime will be installed.

3. In each staging area, click SETUP.EXE to bring up the Oracle Installer. Examine the files listed in the Available Products pane. If you do not want a file to appear in this pane—for example, a file has already been established as a dependency in the VRF file and does not need to be installed explicitly—edit the staging area’s PRD file and change the file’s “Visible?” value to false.

Step 6: Test your installation

Test your installation on a “clean” machine (a machine with no previously-installed products) that is representative of the projected end-user environment. Do not rely on tests conducted on a developer’s machine—that machine may already have files such as icons or libraries that you inadvertently omitted from your map file, or registry settings that were not included in your INS file. This is one of the most common causes of installation problems.

1. Install your application and make sure it installs the components it should.

2. Launch the application to make sure it runs correctly.

3. Test removing your application using the Oracle Installer.

Step 7: Copy the staging area to your distribution media

When you are ready to copy your application to CD, tape, diskette, or another medium—or simply to a LAN or other networked machine—be sure you include the entire staging area—that is, DEPLOY/RELEASE in its entirety. If
you include only your subdirectory, the required runtime environment(s) will not be accessible.
This chapter offers guidelines to help you understand and take advantage of the Developer/2000 Server.

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
</table>
2.1 Understanding the Developer/2000 Server

2.1.1 Introduction to the Developer/2000 Server

The Developer/2000 Server is a new generation of Oracle development tools that enable you to deploy new and existing applications on the World Wide Web, either on an internal company intranet, or on the Internet. The Developer/2000 Server takes advantage of the ease and accessibility of the Web, and elevates it from a static information-publishing mechanism to an environment capable of supporting complex, dynamic applications.

2.1.1.1 The best of the Web and client/server

The Developer/2000 Server provides solutions that enable you to take advantage of all the benefits of the Web, while maintaining the strengths of client/server computing. The Web greatly reduces the costs of administering and maintaining applications, while allowing for a thin, low-cost client. At the same time, it allows you to leverage your existing client/server applications, which may be mission-critical applications that support—and often drive—your business practices. These applications still must be scalable to large numbers of users and open to all of your environments.

2.1.1.2 Three-tiered architecture

In most client/server implementations today, running applications is a highly client-intensive process. Though data is extracted from a remote database server, applications run on client machines, which often have limited processing power and memory capacity.

The Developer/2000 Server supports a three-tiered architecture that delivers the benefits of both client/server and the Web in a single application. In a Web implementation, application logic and processing are focused on a middle tier of application servers instead of on desktop client machines.

<table>
<thead>
<tr>
<th>Tier</th>
<th>Hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>front-end</td>
<td>any number of client desktop machines</td>
</tr>
<tr>
<td>middle</td>
<td>one or more application servers</td>
</tr>
</tbody>
</table>
2.1.2 Benefits of the Developer/2000 Server

The Developer/2000 Server benefits groups across the spectrum of the application life-cycle, including managers, developers, and end users. Each group benefits by gains in productivity, reduced resource requirements, and lower costs:

- **Low-cost deployment and maintenance.** Web applications are deployed and maintained on centralized application servers, from which they download to end users’ Web browsers at runtime. To roll out a Web application, simply give end users the application’s URL. This distribution method reduces the time, cost, and complexity of deploying applications to a large or geographically-dispersed end user base, all without installing application software on their desktop machines.

- **Client/server migration.** The Developer/2000 Server extends and leverages the traditional strength of Oracle’s client/server development suite. Developer/2000 supports all GUI widgets available in a client/server application, so you can Web-deploy any existing client/server application without changing the form definition (.FMB file). You can run the same Forms executable (generated .FMX file) in client/server mode or on the Web.

---

1 See Section 2.3.2.7, “Feature restrictions for forms on the Web”
Programmers experienced with Developer/2000 do not need to learn Java, JavaScript, or any other new language.

- **“Thin client” architecture.** The only client-side requirement is a Java-enabled Web browser. Any end user with a Java-enabled browser can run Web-enabled applications. This greatly reduces processor and memory requirements for end users’ desktop machines.

- **Support for various business objectives.** With the Developer/2000 Server, you can develop internal, transaction-intensive applications (deployed on a company intranet), and electronic commerce applications (deployed on the Internet). End users accustomed to client/server applications will enjoy similar performance with the Developer/2000 Server.

- **NC support.** The Developer/2000 Server is designed in accordance with Oracle’s Network Computing Architecture (NCA). Developer/2000 applications run as NCA-compliant Web cartridges.

- **Platform independence.** The Developer/2000 Server leverages Java—the emerging standard platform-independent language—to map GUI widgets to their native counterparts on other platforms. The look and feel of widgets will differ only slightly between end user platforms (Windows, Mac, Motif, and so on). For more information, see Section 2.3.2.3, “Creating a generic GUI for Web-enabled Form Builder applications”.

- **Web publishing.** Report Builder provides you with a number of features that enable Web publishing of your reports.

- **Web Wizard.** You can quickly Web-enable your existing reports using the Web Wizard in Report Builder. The Web Wizard enables you to specify Web features for the report and immediately generate output in Hypertext Markup Language (HTML) or Portable Display Format (PDF).

- **Drill-down reporting.** You can associate URLs with objects in a report. By associating a URL with an object in a report, you enable end users to easily navigate to other Web pages or launch other reports that provide more detailed or related data.

- **Dynamic reporting.** To dynamically run reports from a Web browser, you can use the Reports Multi-tier Server (hereafter called the Reports Server) in conjunction with the Reports Web Cartridge or Web CGI. When a user clicks on a URL that calls a report, the Reports Cartridge or Web CGI transfers the request to the Reports Server. The Reports Server queues and runs the report.

---

1 Java 1.1.X (see your Developer/2000 release notes)
generating the specified output. The Web server can then serve the report output back to the user in their Web browser.

- **Monitoring the report queue.** Using the Reports Queue Manager, you can administer the Reports Server queue, launch reports, and schedule reports to run automatically at specific times or intervals.

### 2.1.3 Form Builder Web architecture

To run new or existing Form Builder applications on the Web, you should install and operate Developer/2000 on the middle tier of a distributed three-tiered architecture (see Figure 2–1).

The Forms web component of Developer/2000 consists of the *Forms Client* and the *Forms Server.*

#### 2.1.3.1 About the Forms Client

The Forms Client is a Java *applet*—downloaded at runtime from an application server to an end user’s Web browser—that displays the form’s user interface and manages interaction between end users and the Forms Server. The Forms Client receives “bundles” of interface commands from the Forms Server and translates them (in sets) into interface objects for the end user. Some interface events handled by the Forms Server Runtime Engine in a client/server implementation (such as typing characters in a text field, or moving around a dialog) occur only on the Forms Client in the Web implementation, with no interaction with the Forms Server runtime engine.

The Forms Client is:

- **Generic.** You are not required to deploy a separate Java applet for each application you wish to deploy on the Web.

- **Dynamic.** The Forms Client dynamically reacts to the current form at runtime, requesting and displaying only the information and user interface elements necessary to represent the current state of the application at any given time.

- **Feature-rich.** The Forms Client supports all user interface widgets and tools available in a client/server implementation. Due to Java object standards, the look and feel of some Forms widgets may vary slightly when deployed on the Web. For more information, see Section 2.3.2.7, “Feature restrictions for forms on the Web.”

- **Thin.** At startup, only those class files necessary to render the initial state of an application are downloaded to the end user’s machine. Additional class files
are downloaded dynamically (as needed) to support additional user interface functionality.

2.1.3.2 About the Forms Server
The Forms Server consists of two components:

- **Listener.** The Forms Server Listener initiates the Forms Server runtime session and establishes a connection between the Forms Client and the Forms Server Runtime Engine.

- **Runtime Engine.** The Forms Server runtime engine is a modified version of the Forms 5.0 Runtime Engine, with user interface functionality redirected to the Forms Client. It handles all form functionality except UI interaction, including trigger and commit processing, record management, and general database interaction.
2.1.3.2.1 Client/server architecture vs. Web architecture There are two main differences between deploying Form Builder applications in a client/server implementation and a Web implementation:

- **Client/server.** The Forms Server runtime engine (and all application logic) are installed on end users’ desktop machines. Although your application can include database-server-side triggers and logic, typically all user interface and trigger processing occurs on client machines.

![Client-Server Diagram](image-url)

*Figure 2–2 Developer/2000: client/server architecture*
2.1 Understanding the Developer/2000 Server

- **Web.** The Forms Server runtime engine (and all application logic) are installed on application servers, not on client machines. All trigger processing occurs on database and application servers, while user interface processing occurs on the Forms Client.

![Diagram of Developer/2000 for the Web](image)

**Figure 2–3 Developer/2000: Web architecture**

2.1.3.2.2 **Call and response: Forms Client and Forms Server** Once a direct network connection is established between the Forms Client and Forms Server, the two components communicate through a series of requests and responses—via compressed messages passed over a network.

Requests from the Forms Client are events (such as “click button” or “display LOV”). Responses from the Forms Server are a series of changes to the user interface (such as value changes, and adding and removing components), all of which the Forms Client turns into display objects.

For example, the Forms Client might receive a response from the Forms Server similar to “create a green text item on canvas CAN_12.” The Forms Client translates the response into actual interface objects, in this case, the colorful text item.

The Forms Client contacts the Forms Server when users perform:
2.1 Understanding the Developer/2000 Server

- high-level operations (such as accepting or canceling a dialog)
- operations (such as checking a checkbox or navigating between fields) that involve validation processing and cause default and user-defined triggers to fire

2.1.4 Running Form Builder applications on the Web

Developer/2000 for the Web

Figure 2–4 Running Form Builder applications on the Web: process flow

To start and run a Form Builder application on the Web, end users use a Java-enabled Web browser to access a URL. The following sequence occurs automatically:

1. The URL corresponds either to a static (non-cartridge) HTML page, or to an application cartridge, residing on the application server.
2. The HTML page and the Forms Client applet are downloaded from the application server to the user’s browser.

3. The Forms Client sends a request to the Forms Server Listener (which resides on a specific port of the machine from which the Forms Client was downloaded).

4. The Listener contacts the Forms Server runtime engine and connects to a Forms Server runtime process (either by starting a new process, or by connecting to an existing process). If included in the HTML page, Forms command-line parameters (such as form name, user ID and password, database SID, menu name, and so on) and any user-defined Form Builder parameters are passed to the process by the Listener.

5. The Listener establishes a direct socket connection with the Runtime Engine, and sends the socket information to the Forms Client. The Forms Client then establishes a direct socket connection with the Runtime Engine. The Forms Client and Runtime Engine then communicate directly, freeing the Listener to accept startup requests from other end users. The Forms Client displays the application’s user interface in an applet window outside the main window of the end user’s Web browser.

6. As in a client/server implementation, the Runtime Engine communicates directly with the database through SQL*Net (or another driver, for non-Oracle datasources).

2.1.4.1 Security and encryption
Data passed between the database, the Forms Server, and the Forms Client is automatically encrypted before—and decrypted after—transmission by the following protocols:

- **RSA RC4 40-bit encryption** (for transmissions between the Forms Client and the Forms Server)
- **SQL*Net SNS/ANO** (for transmissions between the Forms Server and the database server)

**Note:** Encryption is provided by default, but can be disabled.
2.1.5 Report Builder Web architecture

The Report Builder Web architecture provides a flexible way to publish information on the Web. It enables you to tailor your Web reporting to the needs of your data.

One method of Web publishing is to generate static HTML or PDF output, place the files on a server, and link them to a Web page to make them accessible to users. This method is very useful when you run reports at regular intervals and keep previous versions of the output for sometime. For example, a sales report for the third quarter of the current fiscal year would be generated after the close of the quarter and the output would remain available indefinitely. A highlight report for your department might be published once per week and reports from previous weeks remain available indefinitely.

For other data (e.g., inventory), though, it may be imperative that you see the very latest information available. Dynamic Web reporting generates report output as you navigate to it in your Web browser. When you click a URL, a report is executed and the output is served back to you in your Web browser. This method ensures that you are looking at the very latest information.

Report Builder supports all of these scenarios with its Web Wizard, Report Editor, and Reports Multi-tier Server (hereafter referred to as the Reports Server). The Web Wizard enables you to add basic Web properties (e.g., bookmarks) to an existing report and immediately generate HTML or PDF output. If the output only needs to be generated once, the Web Wizard may be all that you need.

If the output needs to be refreshed at regular intervals (e.g., once per week), you can automate the report’s execution. After using the Web Wizard to add Web properties to the report, you can use the Reports Server to schedule the report to run at regular intervals, regenerating the HTML or PDF output each time.

For dynamic Web reporting, you can use the Reports Server in conjunction with the Reports Web Cartridge or Web CGI. When you click on a link, the URL is processed by the Reports Web Cartridge or Web CGI, and a job request is sent to the Reports Server. The Reports Server sends the job to a runtime engine for execution. When the report output is ready, it can be served back to the Web browser by the Web server.
2.1.5.1 Static Reporting with the Web Wizard
To enable you to quickly prepare existing reports for the Web, Report Builder provides a Web Wizard (accessible from the toolbar and Tools menu). The Web Wizard contains a series of pages that let you specify:

- header and footer HTML files to be added at the beginning and end of the report output
- bookmarks to be added to break columns to create an outline for the report
- the type of output to be generated (HTML, HTML style sheets, or PDF)
- whether to generate output immediately

Note: In addition to the Web Wizard, you can also use the Report Editor to add Web functionality to a report. You should use the Web Wizard to add the basic Web functionality (e.g., bookmarks) and the Report Editor to add more advanced Web functionality (e.g., customized report escapes).

2.1.5.2 Scheduling Reports with the Reports Server
To enable you to schedule reports to run at regular intervals, the Reports Queue Manager and the R30CLI command line provide scheduling options. In the Reports Queue Manager, you can specify scheduling options whenever you launch a new job from the Job menu. On the R30CLI command line, you can use the SCHEDULE argument to specify when to run the report.

2.1.5.3 Dynamic Reporting with the Web Cartridge or CGI
To enable end users to generate reports dynamically from a Web browser, the Reports Server suite includes the following components:

- The Reports Server (R30MTS), which enables you to run reports on a remote server.
- The Reports Web Cartridge (R30OWS), which sends requests from the Web server to the Reports Server if you are using the Oracle Web Application Server.
- The Reports Web Common Gateway Interface (R30CGI), which sends requests from the Web server to the Reports Server using standard CGI.
2.1.5.3.1 About the Reports Server  

The Reports Server enables you to run reports on a remote application server. It handles client requests to run reports by entering all requests into a job queue. When one of the server's runtime engines becomes available, the next job in the queue is dispatched to run. As the number of jobs in the queue increases, the server can start more runtime engines until it reaches the maximum limit specified when the server process was started. Similarly, idle engines are shut down after having been idle for longer than a specified period of time.

You can send job requests to the Reports Server from:

- a Web browser using standard URL syntax (when used with the Reports Web Cartridge or Reports Web CGI). Note that you can also view the output in the Web browser.
- a Form Builder form. Note that you can also view the output in the form.
- the Reports ActiveX control (R30SXA32.OCX), Reports Queue Manager, or R30CLI.

The Reports Server also enables you to:

- Send report output to file, printer, or cache.
- View and manage the report queue in the Reports Queue Manager (R30RQM32.EXE) on Windows and the Reports Queue Viewer (r30rqv) on UNIX.
- Schedule reports using the Reports ActiveX control, Reports Queue Manager, or R30CLI.
- Take advantage of cached report output and duplicate job detection.

For more information on the Reports Queue Manager and Reports ActiveX control, refer to their help systems (r30rqmus.hlp and r30xes32.hlp).

2.1.5.3.2 About the Reports Web Cartridge  

The Reports Web Cartridge (R30OWS) provides an easy connection between the Oracle Web Application Server and the Developer/2000 Reports Server (R30MTS). This connection enables a Web client to run reports on the Reports Server and see the report output on the client Web browser. Like other Web Cartridges, R30OWS is a dynamically-linked, or shared, library that is managed by a Web Request Broker, which is associated with a particular Oracle Web Application Server Listener. R30OWS is an alternative to the Reports Web CGI (R30CGI) and has the same functionality, but it also has native integration with the Oracle Web Application Server.
After URL submission from the client, the Web server invokes the Reports Web Cartridge. The Web Cartridge then does the following:

- Parses the client request.
- Converts it to a command line for the Reports Server.
- Submits the command line to the Reports Server (synchronously).
- After the report is finished, retrieves the name of the report output from the server and creates HTTP redirection (Location:) to the output file. After that, it is up to the Web server to manage this redirection (typically by displaying the file back in the Web client browser).
- Terminates.

An error at any stage causes creation of an HTTP response with HTML content describing the error, and termination of the R30OWS task.

2.1.5.3.3 About the Reports Web CGI The Reports Web CGI (R30CGI) provides a connection between a Web Server and the Reports Server. This connection enables a Web client to run reports and see the report output in the client Web browser. R30CGI is designed to run under WWW CGI v1.1. When the client submits a URL, the Web Server runs the Reports Web CGI.

The Web CGI then does the following:

- Parses the client request.
- Converts it to a command line for the Reports Server.
- Submits the command line to the Reports Server (synchronously).
- After the report is finished, retrieves the name of the report output from the Reports Server and creates HTTP redirection (Location:) to the output file. After that, it is up to the Web Server to manage this redirection (typically by displaying the file back in the Web client browser).
- Terminates.

An error at any stage causes creation of an HTTP response with HTML content describing the error, and termination of the R30CGI task.
To start a report on the Web, end users point to a URL from their Web browsers. The following sequence then occurs:

1. The Web browser passes the URL to the Web server, and the Web server sends the request to the Reports Web Cartridge or CGI.
2. The Reports Web Cartridge or CGI parses the request and converts it to a command line that can be executed by the Reports Server. Then it submits the command line to the specified Reports Server (synchronously).
3. The Reports Server queues the job request from the Web Cartridge or CGI and, when one of its runtime engines becomes available it runs the report.
4. The Reports Web Cartridge or CGI retrieves the name of the report output from the server and creates HTTP redirection to the output file.
5. The Web server executes HTTP redirection, passing the new URL back to the Web browser, and the report is displayed on the client.
2.1.6 Graphics Builder web architecture

The three-tiered architecture of the Developer/2000 Server enables you to enhance your Web publications with interactive, data-driven graphic displays that can be accessed from a remote application server by any client. You can leverage this capability to embed powerful graphical applications directly in HTML pages, or in Form Builder applications on the Web.

2.1.6.1 About the Graphics Client and Graphics Server

The Graphics Builder web architecture consists of three components, as illustrated in Figure 2–6:

- Graphics Client
- Oracle Web Application Server with Web Request Broker component
- Graphics Server

![Graphics Web Architecture](image)

Figure 2–6 Graphics Builder web architecture

2.1.6.2 About the Graphics Client

The Graphics Client displays an interface that resembles the Graphics Runtime interface and allows you to run your existing Graphics Builder applications.
without modification on the Web. Any HTML Web browser running on any platform can function as a Graphics Client.

2.1.6.3 About the Web server with Oracle Web Request Broker module
The Web Request Broker is a separate module provided with Oracle Web Application Server and supported by Netscape, Microsoft Web Server and others. The Web Request Broker provides a framework for, and manages client connections to, cartridges such as the Graphics Server.

2.1.6.4 About the Graphics Server
The Graphics Server is a variation of the Graphics Runtime engine that runs on the Web server and functions as an application server. When invoked by requests or events submitted by the client and relayed through the Web Request Broker, the Graphics Server processes the event, and returns an HTML page containing a reference to an updated display image. The server communicates with the database through SQL*Net (as it does in the traditional client/server implementation).

2.1.6.5 Call and response: Graphics Client and Graphics Server
The Graphics Client and Graphics Server communicate via an HTTP connection through a series of requests and responses relayed through the Web Request Broker. Requests are initiated on the client-side by direct URL submission, mouse-clicks in the display region, or HTML-form submit requests. All requests are passed as one of the following:

- A simple URL
- A parameterized URL containing special keyword arguments similar to those used in Graphics Builder Batch (for example, userids or values for user-defined parameters).

Responses are returned as one of the following:

- An HTML form functioning as a Graphics Builder web interface
- An HTML form containing an image of the requested display

2.1.6.6 About display instances and session IDs
It is possible for multiple instances of a Graphics Builder display to be open simultaneously if:
2.1 Understanding the Developer/2000 Server

- A single user uses more than one browser to open multiple instances of a display at the same time.

- Different users open the same display at the same time.

When multiple instances of a display are open, each instance of the display is unique and independent. That is, when a user performs an action on one instance of a display, other instances of the display are not affected.

The Graphics Server uses session IDs to identify instances of open displays. When the server receives a request for a display, it generates a unique session ID representing the display instance, and embeds the ID in its initial response to the client. The client returns the session ID as a parameter with any event it submits to the server, thus identifying the instance of the display that is being referenced. Note that the generation and submission of session IDs is an automatic process and requires no attention from the end user or developer.

2.1.6.6.1 Accessing Graphics Builder displays via the web interface toolbar

The easiest way to access a Graphics Builder display is through the Graphics Builder web interface toolbar. To use this method, the user points the web browser to the simple URL that invokes the toolbar. For example:

```
http://my_server/ogweb
```

The following sequence then occurs:

1. The client passes the URL to the Web server.
2. The server’s Web Request Broker forwards the request to the Graphics Server.
3. The server returns an HTML-form that functions as the Graphics Builder web interface toolbar.
4. On the toolbar, the user enters a username and password to connect to the database, selects a display from the list of available displays, and clicks Open. This causes the browser to submit a request in the form of a parameterized URL similar to the following:

   ```
   http://my_server/ogweb?openfile=my_display.ogd&userid=scott/tiger@og73&my_parameter=my_value
   ```

5. The Web Request Broker forwards this request to the Graphics Server.
6. The server returns an HTML page containing the requested image.
7. Each time the user executes an event—for example, by clicking the mouse in a display region—the client generates another request. Events are relayed via
automatically-generated parameterized URLs containing keywords describing the details of the event (for example, mouse X and Y positions).

2.1.6.6.2 Accessing Graphics Builder displays via parameterized URLs

Users familiar with the standard keywords used to invoke Graphics Builder Batch from a command line can construct URLs containing the necessary parameters to access Graphic displays directly from the server, rather than through the Graphics Builder web toolbar. This method of accessing a display enables you to:

- Embed Graphics Builder displays in custom-HTML pages.
- Execute PL/SQL code contained in displays.
- Supply additional information (such as userids, or values for user-defined parameters) in display requests.
- Turn the Graphics Builder web interface toolbar on or off.

To accomplish any of the above, the user submits a parameterized URL. For example:

```
http://www.my_company.com/ogweb?openfile=my_display.ogd&userid=scott/tiger@og
73&oracle_interpret="BEGIN; MY-PROC(my_argument); END;"
```

The following sequence then occurs:

1. The client passes the URL to the Web server.
2. The server’s Web Request Broker forwards the request to the Graphics Server.
3. The server performs the operations specified by any additional keywords, and returns an HTML page containing an image of the requested display.
4. Each time the user executes an event—for example, by clicking the mouse in a display region—the client generates another request. Events are relayed via additional, automatically-generated, parameterized URLs containing keywords describing the details of the event (for example, mouse X and Y positions).

For more information about using Graphics Builder command line options in parameterized URLs, see Section 2.1.6.7, “About using Graphics Builder parameters in URLs”.
2.1.6.7 About using Graphics Builder parameters in URLs

There are a number of Graphics Builder keywords that can appear as “name-value” pairs in URLs used to invoke the Graphics Server. For example, a user might submit the following URL:

http://www.myserver.com/ogweb?openfile=mydisplay.ogd&userid=scott/tiger@og73
&showtoolbar=yes

This request employs the `userid`, `openfile`, and `showtoolbar` keywords to connect to a database, open a display, and turn on the Graphics Builder web interface toolbar.

The following table defines the valid keywords and specifies the range of possible values for each:

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Use</th>
<th>Valid values</th>
</tr>
</thead>
<tbody>
<tr>
<td>openfile</td>
<td>Specifies the name of the Graphics Builder display file (.OGD) to open.</td>
<td>The name of any valid Graphics Builder display file. A suffix of .OGD or .OGR is assumed if not specified. For example: openfile=my_display.ogd</td>
</tr>
<tr>
<td>userid</td>
<td>Specifies the full connect string to log on to the desired database.</td>
<td><code>userid/password@dbname</code></td>
</tr>
<tr>
<td>showtoolbar</td>
<td>Turns Graphics Builder web toolbar on or off.</td>
<td>`{yes</td>
</tr>
<tr>
<td>sessionid</td>
<td>The unique identifier for the instance of a Graphics Builder display being viewed via the Graphics Server. For more information about session IDs, see Section 2.1.6.6, “About display instances and session IDs”.</td>
<td>Any alpha-numeric session ID number generated by the Graphics Server. For example: sessionid=000001.091032</td>
</tr>
</tbody>
</table>
### 2.1 Understanding the Developer/2000 Server

#### Deploying Applications on the Web

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Use</th>
<th>Valid values</th>
</tr>
</thead>
</table>
| close   | Closes the specified display. **Note:** This keyword is necessary only where strict control over documents is required for security reasons, since by default the server automatically closes open displays when the period specified in the server parameter **GW_TIMEOUT** has expired. For more information about **GW_TIMEOUT** see Section 2.2.6.2, “Creating the Graphics Server cartridge”. **Note:** All URL submissions to the Graphics Server must use the standard URL format of changing spaces to +, and encoding special characters with %xx hexadecimal encoding. For more information about encoding URLs, see the W3C (World Wide Web Consortium) document “Names and Addresses, URIs, URLs, URNs, URCs” at http://www.w3.org/pub/WWW/Addressing/Addressing.html. | (yes) is the only valid value for the close keyword. For example:  
`sessionid=000001.091032 close=yes` |
2.2 Setting Up the Developer/2000 Server

This section covers these topics:

- Section 2.2.1, “Configuring the Forms Server”
- Section 2.2.4, “Configuring the Reports Server”
- Section 2.2.6, “Configuring the Graphics Server”

2.2.1 Configuring the Forms Server

When you have installed the software onto your application server, you must configure the Developer/2000 Server, your Web server, and Oracle Web Request Broker\(^1\) in order to deploy Form Builder applications on the Web.

For instructions, refer to:

- Step 1: Generating and deploying .FMX files
- Step 2: Starting and stopping the Forms Server Listener
- Step 3: Making applications available to end users
- Step 4: Configuring the Forms Client

\textbf{Note:} If, after installation and configuration, you experience problems running Form Builder applications on the Web, refer to Section 2.2.3, “Troubleshooting your Form Builder configuration”.

2.2.1.1 Step 1: Generating and deploying .FMX files

Once you have developed the form definition (.FMB file) of a form, and you want to deploy the form on the Web, you must do the following:

1. Generate a Forms Server runtime executable (.FMX file).

   You must generate .FMX files on the same platform as the application server on which you will deploy them. For example, if your application server’s operating system is Sun Solaris, you must use the Solaris version of the Forms Generate component to generate .FMX files for deployment on the Web.

2. Deploy the .FMX file on your application server.

   You can deploy .FMX files in any directory on your application server, provided you include the appropriate directory path and filename in the cartridge or HTML file that end users access to run the application.

---

\(^1\) Required only for cartridge implementation.
If you provide only the .FMX filename in your application cartridge or HTML file (with no directory path), the Forms Server runtime engine looks in two places for the file:

- **ORACLE_HOME\BIN\**
  The Runtime Engine searches here first.
- **FORMS50_PATH**
  If the Runtime Engine cannot find the .FMX file in **ORACLE_HOME\BIN\**, it then searches the directory tree defined by the **FORMS50_PATH** environment variable (if defined) for the first occurrence of the .FMX file.

You can view and set the **ORACLE_HOME** and **FORMS50_PATH** environment variables.

### 2.2.1.2 Step 2: Starting and stopping the Forms Server Listener

Before users can run your Web-enabled Form Builder applications, you must start a Forms Server Listener process on your application server. The Listener handles Web application requests from end users.

For instructions, refer to:

- Starting a Forms Server Listener process
- Checking the status of the Forms Server Listener
- Stopping a Forms Server Listener process

#### 2.2.1.2.1 Starting a Forms Server Listener process

On Microsoft Windows NT:

1. On the NT taskbar, choose **Start—>Run**.
2. Type `<ORACLE_HOME>\bin\f50srv32 port=port_number` and press the Enter (or Return) key.
   
   For example: `c:\orant\bin\f50srv32 port=5555`

   A Listener process starts running on the specified port number.
2.2 Setting Up the Developer/2000 Server

On UNIX:

1. From the UNIX prompt, type `cd $ORACLE_HOME/bin` and press the Enter (or Return) key.

2. Type `f50srvm port=port_number &` and press the Enter (or Return) key.
   
   For example: `f50srvm port=4321 &`

   A Listener process starts running (in the background) on the specified port number.

Notes:

- If you do not specify a port number when you start the Listener process, the process starts on port 9000 by default.
- The port number on which you start the Listener process must match the port number you specify in an application’s static (non-cartridge) HTML file, cartridge HTML file, cartridge parameters, or URL. For more information, refer to Section 2.2.1.3, “Step 3: Making applications available to end users”.

2.2.1.2.2 Checking the status of the Forms Server Listener

On Microsoft Windows NT:

1. With your mouse, point to a blank area of the NT taskbar, and display the popup menu.
   
   Note: Most users click the right mouse button to display a popup menu.

2. Choose Task Manager.

3. In the Task Manager, click the Processes tab.
   
   If a Listener process is running, the Task Manager will display a process called `F50SRV32.EXE`, and multiple occurrences of the `F50WEB32.EXE` process (one for every active connection).

On UNIX:

- At the UNIX prompt, type `ps -ef | grep f50` and press the Enter (or Return) key.
   
   A list of process IDs will appear on the screen. If the Listener is running, the list will include a process called `F50SRVM`, and multiple occurrences of the `F50RUNW` process (one for every active connection, plus one spare connection ready for the next user).
2.2.1.2.3 Stopping a Forms Server Listener process

On Microsoft Windows NT:

1. Check the status of the Forms Server Listener.
2. If the Listener is running, the Task Manager will display a process called F50SRV32.EXE.

On UNIX:

1. Check the status of the Forms Server Listener.
   A list of process IDs will appear on the screen. Note the process ID for the f50srvm process.
2. At the UNIX prompt, type kill process_ID and press the Enter (or Return) key.

2.2.1.3 Step 3: Making applications available to end users

Once you have generated and deployed the .FMX files for the Form Builder applications you want to deploy on the Web, make the applications available to end users by following the instructions in:

- Step 3a: Creating virtual directories on your Web server
- Step 3b: Choosing an HTML file construction method, either dynamic or static.
- Step 3c: Providing URLs of your Web-enabled applications

2.2.1.3.1 Step 3a: Creating virtual directories on your Web server

To create pointers to various physical directories on your application server, create virtual directories on your Web server.

For more information about creating virtual directories, refer to your Web Server documentation.

Creating virtual directories offers the following benefits:

- **Simplicity.** When you create an application cartridge or custom HTML file, or use a URL to point to the cartridge or HTML file, you can refer to a virtual directory instead of specifying a lengthy physical directory path.

- **Portability.** If you move files to different directories in the future, you only need to modify the virtual directory settings on your Web server (instead of modifying existing HTML files or cartridges).
Consider creating virtual directories for the following:

- **Applet codebase.** To point to the physical directory where the Forms Client will search for Java class files: `ORACLE_HOME\forms50\java` (for example, `c:\orant\forms50\java`).
  
  **Note:** Do not set the codebase virtual directory to `/ORACLE/`.

- **HTML files.** To point to the physical directory where the Web server will search for static (non-cartridge) HTML files, and cartridge HTML files.

- **JAR files.** To point to the physical directory where the Oracle JAR files (and any custom JAR files) are stored.

Below are examples of virtual directory settings. The directories are provided only as examples, and are not meant as guidelines or requirements, with one exception: the physical directory you define for applet codebase must be `ORACLE_HOME\forms50\java`. Other than applet codebase, choose physical and virtual directory setting according to your own configuration and needs.

<table>
<thead>
<tr>
<th>Purpose</th>
<th>Example Physical Directory</th>
<th>Example Virtual Directory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Applet codebase</td>
<td><code>c:\orant\forms50\java</code></td>
<td><code>/web_code/</code></td>
</tr>
<tr>
<td>HTML files</td>
<td><code>c:\web_forms\html</code></td>
<td><code>/web_html/</code></td>
</tr>
<tr>
<td>JAR files</td>
<td><code>c:\orant\forms50\java</code></td>
<td><code>/web_jars/</code></td>
</tr>
</tbody>
</table>

**Table 2-2 Examples of virtual directories**

### 2.2.1.3.3 Step 3b: Choosing an HTML file construction method

When an end user first starts a Web-enabled Form Builder application (by selecting the application’s URL), an HTML file is downloaded from the application server to the user’s Web browser. The HTML file contains all necessary applet tags, parameters, and parameter values required to run the selected application on the Web.

The initial HTML file can be constructed in two ways:

- **Dynamically.** The initial HTML files is dynamically constructed by the *Forms Cartridge Handler*. The benefit of dynamic HTML file delivery is that you can create a generic cartridge and reuse it (for each of your applications) simply by modifying application URLs.

  To create an application cartridge, you must install the Oracle Web Server on your application server. The Oracle Web Application Server includes a Web
The Request Broker component, which provides a framework for cartridges, and manages client connections to them.

For instructions on implementing dynamically-constructed HTML files, refer to Section 2.2.1.3.4, “Creating an application cartridge”.

- **Statically.** You simply create a static HTML file, hardcoding all information to the individual application. The benefit of static HTML file delivery is that you do not need to install the Oracle Web Application Server on your application server; you can use any web server.

For instructions on implementing static HTML files, refer to Section 2.2.1.3.5, “Creating a static (non-cartridge) HTML file”.

### 2.2.1.3.4 Creating an application cartridge

Create an application cartridge for a Form Builder application by doing the following:

- Creating and registering the cartridge
- Creating a base HTML file for the application cartridge

**Note:** The following instructions are specific to Oracle Web Application Server 3.0. If you are using another Web server, refer to your Web server documentation for information about creating application cartridges.

#### Creating and Registering Your Application Cartridge

1. Install Oracle Web Application Server, if you haven’t already.
   
   **Note:** When you install the Web Application Server, use `WEBSVR` as the name of the default Web Listener (on Windows NT, `WEBSRV` is the default name). You specify the listener name in the Oracle Web Listener Configuration dialog.

2. Start the Admin listener. For example, type: `wlctl30 start admin`.

3. Start the WWW listener. For example, type: `wlctl30 start www`.

4. Start your browser.

5. Access the Oracle Web Application Server by navigating to the appropriate URL (`http://<name of your machine or IP address>:<port number of Web Server Admin listener>`).
   
   For example: `http://myserver.com:9999`.
   
   The Oracle Web Application Server appears.

6. Click **Web Applications Server Manager** to display the Administration page.
7. Click **Oracle Web Application Server**.

8. Click **Cartridge Administration** to display the Cartridge Administration page.

9. Click **Add New Cartridge**.

10. Click **Add New Cartridge with Manual Configuration**, then enter values for the following fields:

<table>
<thead>
<tr>
<th>In this field...</th>
<th>Enter this value...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cartridge Name</td>
<td>The cartridge name. For example: <strong>web_cart</strong></td>
</tr>
<tr>
<td>Object Path</td>
<td>The directory path and filename of the Forms Cartridge Handler.</td>
</tr>
<tr>
<td></td>
<td>- On Microsoft Windows NT: <code>%ORACLE_HOME%/bin\f50webc.dll</code></td>
</tr>
<tr>
<td></td>
<td>- On UNIX: <code>$ORACLE_HOME/lib/f50webc.so</code></td>
</tr>
<tr>
<td>Entry Point</td>
<td><strong>form_entry</strong></td>
</tr>
<tr>
<td></td>
<td><strong>form_entry</strong> is the name of the entry point routine in the cartridge shared library.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> <strong>form_entry</strong> is the required value for Entry Point.</td>
</tr>
<tr>
<td>Minimum # of Instances</td>
<td>The minimum number of users that can simultaneously connect to the cartridge.</td>
</tr>
<tr>
<td>Maximum # of Instances</td>
<td>The maximum number of users that can simultaneously connect to the cartridge.</td>
</tr>
<tr>
<td>Virtual Path (under Virtual Paths)</td>
<td>At runtime, end users pass URLs (HTTP requests) to your Web server. If a request contains the virtual directory associated with your cartridge, the Web server automatically transfers the request to the Web Request Broker. The Web Request Broker in turn starts the cartridge associated with the virtual directory. When defining the Virtual Path, do not include a trailing slash. For example: <code>/web_cart</code></td>
</tr>
<tr>
<td>Physical Path (under Virtual Paths)</td>
<td>The directory path for the Forms Cartridge Handler:</td>
</tr>
<tr>
<td></td>
<td>- On Microsoft Windows NT: <code>%ORACLE_HOME%/bin</code></td>
</tr>
<tr>
<td></td>
<td>- On UNIX: <code>$ORACLE_HOME/lib</code></td>
</tr>
</tbody>
</table>
11. Click Register New Cartridge.
   The Web Application Server registers your cartridge and creates a link for your cartridge in the Web Application Server Home page.

12. Click your cartridge link to display the Cartridge Configuration page.

13. Click Cartridge specific parameters, then enter the following parameters and parameter values:
   
   **Note:** Cartridge parameters and parameter values are case-sensitive.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Parameter Value</th>
</tr>
</thead>
</table>
   | baseHTML       | Enter the physical directory path and filename for the base cartridge HTML file accessed by the Forms Cartridge at runtime. For information on creating a cartridge HTML file, refer to Section , “Creating a base HTML file for the application cartridge”.
   | HTMLdelimiter  | Enter the one-character delimiter used to denote parameter values in the application’s cartridge HTML file. The delimiter can be any character. The default is % (percent sign).

14. (Optional) Enter the following optional cartridge parameters and parameter values.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Parameter Value</th>
</tr>
</thead>
</table>
   | Archive        | Provide the virtual directory path and filename of any JAR file(s) you want downloaded to end users’ Web browsers at application startup. For more information, refer to Section 2.2.1.4.4, “Referencing JAR files”.
   | Code           | Enter the following: oracle.forms.uiClient.v1_4.engine.Main                      |
   | Codebase       | Enter the name of the virtual directory you defined to point to the physical directory ORACLE_HOME\forms50\java\. |
   | serverApp      | Enter the name of the application class you wish to apply to the application. Use application classes for creating application-specific font mapping and icon path settings. For more information, refer to the Section B.1, “About Application Classes”.

Deploying Applications on the Web  2-81
Click Modify Cartridge Configuration.

To activate your changes, do the following:
- Click Listener.
- Click Stop to stop the WEBSVR listener.
- Click Start to restart the WEBSVR listener.

Creating a base HTML file for the application cartridge

To create a base HTML file:

1. Make a copy of CARTRIDG.HTML (an Oracle template file located at ORACLE_HOME/TOOLS/DEVDEM20/WEB).
2. Rename the copy.
   For example: cart.html
3. Open the file in a text editor and modify the text and applet tags as needed.
   For information on the Oracle cartridge HTML file template (including a snapshot, instructions on modifying the template, and examples of customized files), refer to Section A.2, “Working with the Cartridge HTML File Template”.
4. Place the new file in the physical directory that corresponds to the virtual directory you defined for HTML files. **Note:** For information on creating a virtual directory to point to the physical directory where your HTML files are located, refer to Section 2.2.1.3.1, “Step 3a: Creating virtual directories on your Web server”.

---

### Table: Cartridge Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Parameter Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>serverPort</td>
<td>Enter the number of the port on which the Forms Server Listener process was started. In most cases, the port is 9000 (the default). For more information, refer to Section 2.2.1.2, “Step 2: Starting and stopping the Forms Server Listener”.</td>
</tr>
<tr>
<td>&lt;forms_param&gt;</td>
<td>Enter valid Forms command-line parameters (such as module) and corresponding values.</td>
</tr>
<tr>
<td>&lt;user_param&gt;</td>
<td>Enter valid user-defined parameters and corresponding values.</td>
</tr>
</tbody>
</table>
2.2.1.3.5 Creating a static (non-cartridge) HTML file

To deploy non-cartridge Web applications, you must create a static HTML file for each application you deploy on the Web. The easiest way to do this is to modify the static (non-cartridge) HTML file template provided by Oracle.

To create a static HTML file for an application:

1. Make a copy of STATIC.HTML (an Oracle template file located at ORACLE_HOME/TOOLS/DEVDEM20/WEB).

2. Rename the copy.

   For example, if your application is Order Info: order.html

3. Open the file in a text editor and modify the text and applet tags as needed.

   For information on the Oracle static (non-cartridge) HTML file template (including a snapshot, instructions on modifying the template, and examples of customized files), refer to Section A.3, “Working with the Static (Non-Cartridge) HTML File Template”.

4. Place the new file in the physical directory that corresponds to the virtual directory you defined for HTML files. Note: For information on creating a virtual directory to point to the physical directory where your HTML files are located, refer to Section 2.2.1.3.1, “Step 3a: Creating virtual directories on your Web server”.

2.2.1.3.6 Step 3c: Providing URLs of your Web-enabled applications

Once you have created a cartridge or static HTML file for an application, and have deployed the corresponding .FMX files, you must give end users access to the application. To do this, simply provide end users with the URL for the application cartridge or HTML page. End users then contact the URL with their Java-enabled Web browser and run the corresponding application.

Note: Another idea is to create an application menu that contains URL links to all of your Web-enabled Form Builder applications. For more information, see Section 2.3.1.3, “Creating an HTML application menu”.

The URL you create will differ depending on whether your application is a cartridge or non-cartridge implementation.
2.2 Setting Up the Developer/2000 Server

Cartridge URL

If you created a cartridge for your application, the URL you give to end users must contain the virtual directory associated with the application cartridge, plus parameter values (if required). For example, to announce the availability of its new cartridge-driven Inventory Checking application, General Traders, Inc. gives the following URL to their end users:

http://gti.com/web_cart?module=inv&userid=clerk@stock&deptid=90

GTI’s URL consists of the following components:

- Protocol: http
- Domain: gti.com
- Web server listener port: 80 (default, implicit)
- Cartridge virtual directory: /web_cart
- Command-line parameters: module (value = inv) and userid (value = clerk@stock)
- User-defined parameter: deptid (value = 90)

Non-cartridge URL

If you created a static HTML page for your application, then the URL you give to end users simply points to the page. For example, to announce the availability of its new Order Tracking application, ABC Corp. might broadcast the following URL:

http://www.abc.com:6666/apps_html/order.html

ABC’s URL consists of the following components:

- Protocol: http
- Domain: www.abc.com
- Web server listener port: 6666 (explicit)
- HTML files virtual directory: /apps_html
- Static HTML file: order.html
2.2.1.4 Step 4: Configuring the Forms Client

When an end user starts up a Web-enabled Form Builder application, the Forms Client (and related Java class files) download from the application server to the user’s browser. As the user continues interacting with the application, additional Java class files are downloaded on an as-needed basis.

You can control how class files are downloaded to the user’s browser. There are two methods:

- **Incremental.** If you select incremental download (the default), only those class files required to render the initial state of the application download at startup. For example, if the initial state of an application includes only a text field and a button, then three Java class files download immediately:
  
  1. Runform (the Forms Client, downloaded first by default)
  2. CfmsTextfield (the class file for text fields)
  3. CfmsButton (the class file for buttons)

  If the end user clicks the button to display an alert, the class file CfmsAlert downloads to the client. Note that a network roundtrip is required for each class file downloaded to the user’s machine.

- **Bundled.** With a bundled download, one or more “bundles” of class files are downloaded to the client machine at application startup. The advantage of bundled download is that each bundle downloads in a single network roundtrip. To download class files in bundles, you must reference a JAR file in the application’s cartridge settings or HTML file.

2.2.1.4.1 About JAR files JAR files are used for aggregating multiple files (Java class files, images, and so on) into one file. Using JAR files increases the speed with which the Forms Client applet and related class files download (and start executing) for two reasons:

- A JAR file downloads in a single HTTP transaction, rather than opening a new connection for each component file.
- JAR file format supports compression, which reduces the size of the downloaded file(s).

At application startup, when Java identifies the JAR in the HTML file, the JAR downloads to the browser and separates into its component files. When the Forms Client requests a new class file during execution, it searches for it in the archives (class files from JAR files) associated with the applet. This search occurs on the client machine. If the applet does not find the class file in the archives on the client
machine, it will search for the class file on the application server from which the applet was downloaded.

### 2.2.1.4.2 About the generic Oracle JAR file

The Oracle Installer automatically installs a generic JAR file (`ORACLE_HOME/forms50/java/formsweb.jar`), that contains many commonly-used Form Builder and message class files. The `formsweb.jar` file includes `Runform.class`, which is the first class file to be executed at application startup.

### 2.2.1.4.3 Creating custom JAR files

If the collection of class files in the generic Oracle JAR file does not suit your needs, you can create your own JAR file.

To create a custom JAR file, you must use the JAR utility from Javasoft to create bundles of Form Builder Java class files. To get the JAR utility and the documentation explaining how to use it, contact Javasoft's website at:

www.javasoft.com/

If you create your own JAR files, be sure to:

- locate your JAR file(s) in a physical directory outside of the `ORACLE_HOME` directory tree.
- create a virtual directory on your Web server to point to the physical directory where your JAR files are located.
- include `Runform.class` inside the JAR file, since it is the first class file executed at application startup. If Java cannot find `Runform.class` in the class files separated out of your custom JAR file, it will search for `Runform.class` on the application server. This extra network roundtrip defeats the purpose of JAR files.

### 2.2.1.4.4 Referencing JAR files

To reference a JAR file to be downloaded at startup, simply add the `ARCHIVES` tag to the parameter settings of the application cartridge or HTML file.

**Notes:**

- If you reference the Oracle JAR file, the virtual directory path of the JAR file must be the virtual directory defined for the applet codebase.
- If you reference custom JAR files, the virtual directory path of the JAR files must be the virtual directory defined for custom JAR files.
- You can reference multiple JAR files in a single `ARCHIVES` tag. Simple separate each JAR file reference with a “+” (addition sign).
The order in which multiple JAR files are downloaded is determined by the order (left to right) in which they are referenced after the `ARCHIVES` tag.

### 2.2.1.4.5 Example of adding the ARCHIVES tag to an application’s static HTML file:

```xml
<applet codebase="/web_code/"
        code="oracle.forms.uiClient.vl_4.engine.Runform">
  <param name="archives" value="/..\web_code\formsweb.jar + \
                      /..\acme_jars\basic.jar + \
                      /..\acme_jars\advanced.jar">
  <param name="serverPort" value="9000">
  <param name="serverArgs" value="module=orders">
</applet>
```

### 2.2.2 Setting up load balancing

The load balancing feature allows you to dynamically balance the load of multiple Forms Servers across systems. Load balancing directs a Forms Client to connect to a Forms Server running on the least-loaded system that is available. Which system is least loaded is determined by a count of the total number of processes running on that system.

Following is an example of a possible load balancing configuration.
The Metrics Server and Metrics Clients are services on Windows NT and they are daemons on UNIX. The software for these services is included when you install Form Builder. The Metrics Server and Metrics Clients do not become active until you complete the steps listed in Table 2–1, “Load balancing roadmap”.

* The Forms Cartridge asks the Metrics Server for the name of the least-loaded system. Then the Forms Cartridge creates an HTML file, including the host name, and sends it back to an end user’s Web browser.
When you use load balancing, the following events occur:

1. Metrics Clients periodically send load information to the Metrics Server. This load information is based on the total number of processes running on each Metrics Client.

2. End users access a URL pointing to a Forms Cartridge.

3. The Forms cartridge asks the Metrics Server for the name of the least-loaded system that is available.

4. The Forms cartridge dynamically creates an HTML file with the name of the least-loaded system specified as the system on which to run the Forms Server and returns that HTML file to an end user’s Web browser.

5. The user’s Web browser then requests the Java applet to be downloaded from the host specified in the HTML file.

6. The Java applet sends a request to the Forms Server Listener asking for a particular Form Builder application (that is, an FMX).

7. The listener contacts a Forms Server Runtime Engine. (The listener maintains a pool of available Runtime Engines to minimize application startup delays.) Each active user receives a dedicated Runtime Engine.

8. The listener establishes a direct socket connection with the Runtime Engine, and sends the socket information to the Java applet. The Java applet then establishes a direct socket connection with the Runtime Engine. The Java applet and the Runtime Engine now communicate directly, freeing the listener to accept startup requests from other end users. (At this point, neither the Web server nor the Forms listener is involved in the communication between the applet and the Runtime Engine.) The Java applet displays the application’s user interface in an applet window outside the main window of the end user’s Web browser.

9. As in a client-server implementation, the Runtime Engine communicates directly with the database through SQL*Net or ODBC, depending on the data source.

Metrics Clients continue to send load information to the Metrics Server and all new service requests are routed based on that information.

Note: If the Metrics Server is unavailable, at Step 3 the Forms cartridge will not get any information back about which is the least-loaded system. The Forms cartridge will then, instead, specify the value of the parameter MetricsServerErrorURL for the end user’s Web browser to request. The user does not necessarily know this is happening as the redirect is behind the scenes from the user’s viewpoint. Refer to
Section 2.2.2.1, “Configuring your Web cartridge for load balancing”, for information about the MetricsServerErrorURL parameter.

Requirements
Following are the requirements for using the Forms Server load balancing feature:

- Metrics Server on one system
  - Forms Server (includes the Metrics Server and the Forms cartridge)
    Note: All Forms Servers must listen on the same port number on each of the systems.
  - Oracle WebServer (release 2.1) or Web Application Server (release 3.0)

- Metrics Clients on all systems running Forms Server
  - Forms Server (includes the Metrics Client)
    Note: All Forms Servers must listen on the same port number on each of the systems.
  - Any Web server
Roadmap for using your load balancing environment

Following is the sequence of steps to take to set up and use your load balancing environment:

<table>
<thead>
<tr>
<th>Step</th>
<th>Refer to...</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Configure your load balancing Web cartridge.</td>
</tr>
<tr>
<td>2.</td>
<td>Install your Metrics Server and Metrics Client.</td>
</tr>
<tr>
<td>3.</td>
<td>Modify your cartridge base HTML file.</td>
</tr>
<tr>
<td>4.</td>
<td>Start your Metrics Server.</td>
</tr>
<tr>
<td>5.</td>
<td>Start your Oracle Web Server (on the same system as your Metrics Server).</td>
</tr>
<tr>
<td>6.</td>
<td>Start your Metrics Clients.</td>
</tr>
<tr>
<td>7.</td>
<td>Start the httpd/Web listener on each Metrics Client system.</td>
</tr>
</tbody>
</table>

The following sections describe the steps for installing, configuring, and setting up load balancing for your Forms Server.

### 2.2.2.1 Configuring your Web cartridge for load balancing

First do the steps in Section 2.2.1.3.4, “Creating an application cartridge”.

Then:

1. Access the Oracle Web Application Server by navigating to the appropriate URL (http://<name of your machine or IP address>:<port number of Web Server Admin listener>).
   For example: http://myserver.com:9999.
   The Oracle Web Application Server appears.

2. Click **Web Applications Server Manager** to display the Administration page.

3. Click **Oracle Web Application Server**.

4. Click **Cartridge Administration** to display the Cartridge Administration page.
5. Click your Forms Web cartridge.

6. Click **Cartridge specific parameters**, and then enter the following parameters and parameter values which are used by the Forms Server for load balancing:

   **Note:** Cartridge parameters and parameter values are case sensitive.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Parameter Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MetricsServerHost</td>
<td>Enter the name of the host (system) where the Metrics Server is running. This parameter is required if load balancing is used. For example: system1.company.com</td>
</tr>
<tr>
<td>MetricsServerPort</td>
<td>Enter the port number on which the Metrics Server will be listening for requests from the cartridge for the least-loaded system. This parameter is required if load balancing is used and its value must be different from the Forms Listener value. For example: 70000</td>
</tr>
<tr>
<td>MetricsServerErrorURL</td>
<td>Enter the URL to which the Web Client will be directed if the Metrics Server does not respond with the name of the least-loaded system within the MetricsTimeout period. This parameter is required if load balancing is used. The URL can:</td>
</tr>
<tr>
<td></td>
<td>■ Point to a Forms Web cartridge on another host, which will then automatically service the request. For example: <a href="http://altsys.company.com/web_cart?module=inventory&amp;userid=clerk&amp;deptid=90">http://altsys.company.com/web_cart?module=inventory&amp;userid=clerk&amp;deptid=90</a></td>
</tr>
<tr>
<td></td>
<td>■ Point to a static or dynamic HTML page on any Web server. This page can be used for giving help and instructions to users when the Metrics Server goes down and load balancing is disabled. For example: <a href="http://altsys.company.com/apps_html/help.html">http://altsys.company.com/apps_html/help.html</a></td>
</tr>
</tbody>
</table>
7. Click **Modify Cartridge Configuration**.

8. To activate your changes, do the following:
   1. Click **Listener**.
   2. Click **Stop** to stop the WEBSVR listener.
   3. Click **Start** to restart the WEBSVR listener.

   **Note:** If you plan to use any of your Metrics Clients as backup Metrics Servers, then you need to do these steps for those Metrics Clients as well. Otherwise, you only need to do these configuration steps for the Metrics Server.

### 2.2.2.2 Installing the Metrics Server and Clients

Following are the steps to install the Metrics Server and Metrics Clients.

#### Installing the Metrics Server

On Windows NT, run the following command to install the Metrics Server as an NT Service:

```
d2ls1032 -install
```

To deinstall the Metrics Server service:

```
d2ls1032 -remove
```

On UNIX, the Metrics Server runs as a daemon so there is no installation required. The name of the Metrics Server executable on UNIX is `d2ls10`. 

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Parameter Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MetricsTimeout</td>
<td>(Optional) Enter the timeout value, in seconds, for the Forms cartridge to use while making requests to the Metrics Server for the name of the least-loaded system. If the cartridge does not receive a response in that time interval, it replaces the %LEASTLOADEDHOST% entry in the HTML file with the MetricsServerErrorURL value. The MetricsTimeout parameter is optional. If not specified, the default value is 30 seconds. For example: 35</td>
</tr>
</tbody>
</table>
Installing Metrics Clients
On Windows NT, run the following command on each system to install each
Metrics Client as an NT Service:

   d2lc1032 -install

To deinstall the Metrics Client service:

   d2lc1032 -remove

On UNIX, the Metrics Client runs as a daemon so there is no installation required.
The name of the Metrics Client executable on UNIX is d2lc10.

2.2.2.3 Modifying your cartridge base HTML file
%LEASTLOADEDHOST% is a fixed-name place holder in the base HTML file. This
place holder is replaced dynamically by the cartridge with the name of the
least-loaded system. The cartridge gets this information from the Metrics Server at
runtime.

Refer to Section , “Creating a base HTML file for the application cartridge”, for
information about creating your base HTML file.

Refer to Section A.2.2, “Modifying the cartridge HTML file template”, for
information about using the %LEASTLOADEDHOST% parameter in your base
HTML file.

2.2.2.4 Starting the Metrics Server

Starting the Metrics Server on Windows NT
1. Click Start—>Settings—>Control Panel.
2. Double-click Services.
4. Enter values for the following startup parameters:

   MetricsClientToServerPort# MetricsServerToCartridgePort#
   [max_no_MetricsClients]

where:
   ■ MetricsClientToServerPort# is the port on the Metrics Server where the
     Metrics Clients will connect and will send their load information.
2.2 Setting Up the Developer/2000 Server

2.2.2 Setting Up the Developer/2000 Server

2.2.2.5 Starting the Metrics Clients

Do these steps for each system where a Metrics Client will run.

Starting the Metrics Clients on Windows NT

1. Click Start—>Settings—>Control Panel.
2. Double-click Services.
3. Choose D2LC Client Service.
4. Enter values for the following startup parameters:

```
MetricsServerHostName MetricsClientToServerPort# MetricsClientLocalPort# [ScaleFactor]
```

5. Click Start to start the service.

Starting the Metrics Server on UNIX

Enter the following command to start your Metrics Server on UNIX:

```
d2ls10 MetricsClientToServerPort# MetricsServerToCartridgePort# [max_no_MetricsClients]
```

where:

- **MetricsClientToServerPort#** is the port on the Metrics Server where the Metrics Clients will connect and will send their load information.
- **MetricsServerToCartridgePort#** is the port on the Metrics Server which the cartridges will use to query the Metrics Server for information.
- **max_no_MetricsClients** is the maximum number of Metrics Clients that will be running and sending load information to the Metrics Server. This parameter is optional. The default value is 1000.
where:

- **MetricsServerHostName** is the name of the system where the Metrics Server is running.
- **MetricsClientToServerPort#** is the port on the Metrics Server where the Metrics Clients will connect.
- **MetricsClientLocalPort#** is the port on the Metrics Client which the Client will use to query the Metrics Server for information.
- **ScaleFactor** is an optional parameter that allows you to reduce the imbalances resulting from varying capacities of Metrics Clients. Since Forms Server load balancing is based only on total number of processes running on each Metrics Client, a system that appears to be the least-loaded system may not necessarily be the best place to run a new process. You should assign a higher value for **ScaleFactor** for your lower capacity systems. The default value for **ScaleFactor** is 1 for UNIX and 4 for Windows NT.

5. Click **Start** to start the service.

**Starting the Metrics Clients on UNIX**

Enter the following command to start each Metrics Client that will run on UNIX:

```
d2lc10 MetricsServerHostName MetricsClientToServerPort# MetricsClientLocalPort# [ScaleFactor]
```

where:

- **MetricsServerHostName** is the name of the system where the Metrics Server is running.
- **MetricsClientToServerPort#** is the port on the Metrics Server where the Metrics Clients will connect.
- **MetricsClientLocalPort#** is the port on the Metrics Client which the Client will use to query the Metrics Server for information.
- **ScaleFactor** is an optional parameter that allows you to reduce the imbalances resulting from varying capacities of Metrics Clients. Since Forms Server load balancing is based only on total number of processes running on each Metrics Client, a system that appears to be the least-loaded system may not necessarily be the best place to run a new process. You should assign a higher value for **ScaleFactor** for your lower capacity systems. The default value for **ScaleFactor** is 1 for UNIX and 4 for Windows NT.
2.2 Setting Up the Developer/2000 Server

Refer to Section 2.2.1.3.6, “Step 3c: Providing URLs of your Web-enabled applications”, for information about giving end users access to the application.

2.2.3 Troubleshooting your Form Builder configuration

If you experience trouble running Form Builder on the Web, check the table below for common symptoms and remedies:

<table>
<thead>
<tr>
<th>Problem</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>You cannot run Web-enabled Form Builder applications with a non-Java-enabled Web browser.</td>
<td>If you are not sure your Web browser is Java-enabled, check the network preferences of your Web browser. The Enable Java and Enable JavaScripts check boxes must be set to checked.</td>
</tr>
<tr>
<td>You see the error message Cannot bind to port 9000 when you try to start the Forms Server.</td>
<td>It is likely that another process is using the port. It could be another occurrence of the Forms Server, so check that it is not already running (see Section 2.2.1.2, “Step 2: Starting and stopping the Forms Server Listener”). If you just stopped the Forms Server, it may take a minute or two for existing connections to port 9000 to reopen.</td>
</tr>
<tr>
<td>The Forms Client does not download to your Web browser.</td>
<td>Check that you have not defined a virtual directory called /ORACLE/ to point to the Oracle Java class files (codebase).</td>
</tr>
</tbody>
</table>
### Problem
The HTML page and applet download at startup, and the applet starts running but nothing else seems to happen.

### Solution
Check the following:
- First, check that the Forms Client indeed is running; if it is, you should see a message in the status bar of your Web browser: `applet oracle.forms.uiClient_v1_4.engine.Runform running`

If you see this message, but your application still does not appear, check the following:

1. Make sure the Forms Server and your Web server are installed on the same application server. Due to a current Java restriction, they must be installed on the same server.

2. Check your application cartridge or HTML file to make sure you specified a valid directory path and filename for the `.FMX` file. You must use a physical directory path, not a virtual directory path.

3. Try setting a preference in your Web browser to display the Java console. This allows you to view runtime Java error messages.

### Problem
The Forms Server seems to ignore the user ID, password, and database SID parameter values you pass in your application cartridge or HTML file.

### Solution
Make sure you preface the values with the parameter `userid=`. For example: `userid=scott/tiger@inventory`.

### Problem
The Forms Server seems to not pick up your variable changes.

### Solution
Stop and restart the Forms Server Listener. For more information, see Section 2.2.1.2, “Step 2: Starting and stopping the Forms Server Listener”.

### Problem
You experience problems when using a security firewall, and you are using a proxy server to access anything outside the firewall.

### Solution
Make sure your proxies are set to manual configuration.
2.2 Setting Up the Developer/2000 Server

2.2.4 Configuring the Reports Server

When you install Developer/2000 on your application server, the Reports Server, Web Cartridge, and Web CGI, are copied along with other required files. For dynamic Web reporting, you must configure the Reports Server and either the Reports Web Cartridge or Web CGI.

**Note:** If you are not doing dynamic Web reporting, you can simply configure the Reports Server by itself, without the Reports Web Cartridge or Web CGI.

This section covers these topics:

<table>
<thead>
<tr>
<th>Problem</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>You experience trouble connecting to a local database.</td>
<td>It could be a result of the following:</td>
</tr>
<tr>
<td>■ If you do not specify a SQL*Net v2 connect string, you will receive errors. The Forms Server runtime engine will not accept connect strings of type LOCAL, TWO_TASK, and so on.</td>
<td></td>
</tr>
<tr>
<td>■ If you are using a SQL*Net v2 connect string and you still cannot connect to the database, make sure the Forms Server Listener is running; on most installations, the Listener is not automatically restarted after a reboot. For information on starting the Listener, see Section 2.2.1.2.1, “Starting a Forms Server Listener process”.</td>
<td></td>
</tr>
<tr>
<td>■ You must have the valid connect string in the TNSNAMES.ORA file on your application server, not on your client machine. The application logic is running on an application server, not on end users client machines.</td>
<td></td>
</tr>
</tbody>
</table>

| You experience unpredictable behavior after modifying the CLASSPATH environment variable | Changing the setting of the CLASSPATH environment variable—on your application server or on an end user’s machine—can produce unpredictable results. Setting the variable to a directory that overlaps with the directory tree where Forms Java class files are located can cause filename overlap. |

| There appears to be several unused processes running on the server | Recall that for each user running a Web-enabled Form Builder application, a Forms Server runtime process (f50run32.exe or f50runw) starts on your application server. Each runtime process should end when the user exits the application, but will remain on the server if a user exits the browser without cleanly exiting the application. To cleanly exit the application, use the menu or the [Exit/Cancel] key function, then exit the browser. |
2.2 Setting Up the Developer/2000 Server

- Section 2.2.4.1, “Step 1: Setting up the Reports Server”
- Section 2.2.4.2, “Step 2: Setting up the Reports Web Cartridge or Web CGI”
- Section 2.2.4.3, “Step 3: Specifying URL run requests”
- Section 2.2.4.4, “Step 4 (Optional): Using the URL mapping feature”
- Section 2.2.4.5, “Step 5: Deploying reports on the Web”

2.2.4.1 Step 1: Setting up the Reports Server

The setup of the Reports Server varies between platforms. The sections that follow describe the setup on each major platform. The online document R30MTSUS.HTM also contains information on setting up and using the Reports Server.

2.2.4.1.1 Setting up for Windows NT as a service

On Windows NT, the Reports Server is best run as an NT service. As with any other service, you can configure the Reports Server service to be started manually or automatically when the system is brought up.

1. Install Report Builder in its entirety or just the Reports Server, Web Cartridge, and Web CGI.

2. Add a line of the following form to

$ORACLE_HOME\network\admin\tnsnames.ora:

<tnsname> = (ADDRESS=(PROTOCOL=tcp)(HOST=<hostname>)(PORT=<portnum>))

where <tnsname> is the name of the Reports Server instance. If you are using a sqlnet.ora file and default_domain is specified, remember to fully qualify the Reports Server TNS name with the default domain. <hostname> is the IP address of the machine. <portnum> is the port number to which the Reports Server is listening. Below is an example entry in tnsnames.ora:

repserver.world=(ADDRESS=(PROTOCOL=tcp)(HOST=12.34.56.78)(PORT=1949))

3. Install the Reports Server as an NT service by running the following command line:

r30mts32 -install <tnsname> tcpip

Following <tnsname> are the names of NT services upon which the Reports Server depends (e.g., tcpip).

4. Choose Start → Settings → Control Panel. Double-click Services. Look for the entry “Developer/2000 Reports Multi-tier Server for <tnsname>.” Click Startup. From this startup dialog, select “This Account” on the Log On As section and type in your operating system username and password. This
specifies that the Reports Server is run as you. In addition, you can also set the service to be started automatically (when the system is booted up) or manually.

5. To start the service, choose Start → Control Panel → Services, select the service, optionally enter command line arguments in Startup Parameters, and click on the Start button. Command line arguments include:

```
name=<tnsname> [minengine=<number> maxengine=<number>
initengine=<number> maxidle=<number>]
```

6. A configuration file is created when the Reports Server is first started up. The file is $ORACLE_HOME\report30\server\<tnsname>.ora. Note that if you manually edit the file, the changes are not picked up by the Reports Server until you shut it down and restart it.

For more information, refer to Section 2.2.4.1.5, “Reports Server configuration file”.

Usage Notes

- To run the server in a Web environment, you need to set up the Reports Web Cartridge or Web CGI.
- To enable reports to be run from Oracle InterOffice, ensure that the RWICLI environment variable points to the thin client executable for your platform, R30CLI32.EXE on Windows 95/NT and r30clim on UNIX platforms. For example:
  
  `C:\ORACLE\BIN\r30cl132.exe`

- If you alter environment variable values (e.g., REPORTS30_PATH) while the Reports Server is running, you must shut down and restart the Reports Server to pick up the changes.

- You can run multiple Reports Servers on one machine. Just make sure that you specify different port numbers.

- When the Server is started up, the configuration file will be read, if it exists, or created, if it does not exist. The four optional server command line arguments (minengine, maxengine, initengine, and maxidle) can be specified in the Startup Parameters in the Control Panel Services window to override the settings in the configuration file. For more information, refer to Section 2.2.4.1.5, “Reports Server configuration file”.

- Make sure printers are accessible by the service.

- When running the Reports Server as a service, there is a known problem with Windows NT not sending report output to mail (DESTYPE=MAIL). Microsoft plans to fix this in Microsoft Exchange Server 5.0. In the meantime, you can
solve this problem by running the Reports Server in the foreground by running the following command: R30MTS32.EXE -listen <tnsname>.

- If the service is installed to be run under a user other than SYSTEM, make sure that the user account has the Password Never Expires option selected in the User Manager and has membership in the appropriate groups to run the Reports Server and access the report files. Also make sure that the user can log on to a service. Choose Start → Programs → Administrative Tools → User Manager, then Policies → User Rights. Check Show Advanced User Rights. From the Right list, choose Log on as a service. If the user is not already in the Grant To list, click the Add button and add the user. Also make sure the default printer is set and the user has at least print permission on the printer.

- Note that by default, the SYSTEM user does not have access to printers. In order for the Reports Server to run properly, it must have access to printers. Given this, you must either set up a separate user to run the Reports Server or give the SYSTEM user access to printers.

- If the service is installed to be run under the SYSTEM user, an icon will appear on the console desktop.

- Since network drives are mapped to a drive letter on a per-user basis, these mappings are no longer in effect when the NT user logs off. The Reports Server must not refer to these drives through their drive letters. Instead you should use UNC pathnames (e.g., \sales\documents\reports). This applies to Reports Server parameters, CGI command mappings, and each hard-coded path name in each report being run.

- To uninstall the Reports Server service, Start → Settings → Control Panel. Double-click Services, select the service, and click Stop. Then run the following command line:

  r30mts32 -uninstall <tnsname>
2.2.4.1.2 Setting up for Windows NT as a non-service In general, it is best to install the Reports Server as an NT service, but it can also be installed as a non-service.

1. Install Report Builder in its entirety or just the Reports Server, Web Cartridge, and Web CGI.

2. Add a line of the following form to

   $ORACLE_HOME\network\admin\tnsnames.ora:

   <tnsname> = (ADDRESS=(PROTOCOL=tcp)(HOST=<hostname>)(PORT=<portnum>))

   where <tnsname> is the name of the Reports Server instance. If you are using a sqlnet.ora file and default_domain is specified, remember to fully qualify the Reports Server TNS name with the default domain. <hostname> is the IP address of the machine. <portnum> is the port number to which the Reports Server is listening. Below is an example entry in tnsnames.ora:

   repserver.world = (ADDRESS=(PROTOCOL=tcp)(HOST=12.34.56.78)(PORT=1949))

3. Run the following command line to start the Reports Server:

   r30mts32 -listen name=<tnsname> [minengine=<number> maxengine=<number> initengine=<number> maxidle=<number>]

   For more information, refer to Section 2.2.4.1.5, “Reports Server configuration file”.

Usage Notes

■ To run the server in a Web environment, you need to set up the Reports Web Cartridge or Web CGI.

■ To enable reports to be run from Oracle InterOffice, ensure that the RWICLI environment variable points to the thin client executable for your platform, R30CLI32.EXE on Windows 95/NT and r30clim on UNIX platforms. For example:

   C:\ORANT\BIN\r30cli32.exe

■ If you alter environment variable values (e.g., REPORTS30_PATH) while the Reports Server is running, you must shut down and restart the Reports Server to pick up the changes.

2.2.4.1.3 Setting up for Windows 95

1. Install Report Builder in its entirety or just the Reports Server, Web Cartridge, and Web CGI.

2. Add a line of the following form to

   $ORACLE_HOME\network\admin\tnsnames.ora:
2.2 Setting Up the Developer/2000 Server

<tnsname> = (ADDRESS=(PROTOCOL=tcp) (HOST=<hostname>) (PORT=<portnum>))
where <tnsname> is the name of the Reports Server instance. If you are using
a sqlnet.ora file and default_domain is specified, remember to fully
qualify the Reports Server TNS name with the default domain. <hostname>
is the IP address of the machine. <portnum> is the port number to which the
Reports Server is listening. Below is an example entry in tnsnames.ora:
repserver.world=(ADDRESS=(PROTOCOL=tcp) (HOST=12.34.56.78) (PORT=1949))

3. Run the following command line to start the Reports Server:
r30mts32 name=<tnsname> [minengine=<number> maxengine=<number>
initengine=<number> maxidle=<number>]
For more information, refer to Section 2.2.4.1.5, “Reports Server configuration
file”.

Usage Notes
- To run the server in a Web environment, you need to set up the Reports Web
  CGI or cartridge.
- To enable reports to be run from Oracle InterOffice, ensure that the RWICLI
  environment variable points to the thin client executable for your platform,
  R30CLI32.EXE on Windows 95/NT and r30clim on UNIX platforms.
  C:\ORACLE\BIN\r30cli32.exe
- If you alter environment variable values (e.g., REPORTS30_PATH) while the
  Reports Server is running, you must shut down and restart the Reports Server
to pick up the changes.
- When the Reports Server is running, an icon will appear in the notification area
  in the task bar. Double-clicking the icon displays a window showing the basic
  server status.
- You should not log off of Windows 95 when the Reports Server is running.

2.2.4.1.4 Setting up for UNIX

1. Install Report Builder in its entirety or just the Reports Server, Web Cartridge,
   and Web CGI.
2. Add $ORACLE_HOME/bin to the PATH environment variable.
3. Set TNS_ADMIN to the location of tnsnames.ora (e.g.,
   $ORACLE_HOME/network/admin).
4. Add a line of the following form to
   $ORACLE_HOME/network/admin/tnsnames.ora:
<tnsname> = (ADDRESS=(PROTOCOL=tcp)(HOST=<hostname>)(PORT=<portnum>))

where <tnsname> is the name of the Reports Server instance. If you are using a sqlnet.ora file and default_domain is specified, remember to fully qualify the Reports Server TNS name with the default domain. <hostname> is the IP address of the machine. <portnum> is the port number to which the Reports Server is listening. Below is an example entry in tnsnames.ora:

repserver.world=(ADDRESS=(PROTOCOL=tcp)(HOST=12.34.56.78)(PORT=1949))

5. Run the following command line to the background to start the Reports Server:

r30mtsm name=<tnsname> [minengine=<number> maxengine=<number> initengine=<number> maxidle=<number>]

For more information, refer to Section 2.2.4.1.5, “Reports Server configuration file”.

Usage Notes

■ To run the server in a Web environment, you need to set up the Reports Web Cartridge or Web CGI.

■ To enable reports to be run from Oracle InterOffice, ensure that the RWICLI environment variable points to the thin client executable for your platform. R30CLI32.EXE on Windows 95/NT and r30clim on UNIX platforms. For example:

/oracle/bin/r30clim

■ If you alter environment variable values (e.g., REPORTS30_PATH) while the Reports Server is running, you must shut down and restart the Reports Server to pick up the changes.

2.2.4.1.5 Reports Server configuration file

The Reports Server configuration file specifies the parameters for the Reports Server. When the Reports Server is started, it gets a TNS name to listen to from the installation or the command line. The server then looks for a configuration file with the TNS name as its filename and .ora as its file extension in $ORACLE_HOME\REPORT30\SERVER. If the configuration file is not present, a default one is created.

Note: Because of its importance, you should restrict access to the configuration file.

The format of the configuration file is:

identifier=string
maxconnect=number
sourcedir=path
2.2 Setting Up the Developer/2000 Server

cachedir=directory
tempdir=directory
cachesize=number
minengine=number
maxengine=number
initengine=number
maxidle=number
security=number
englife=number

Usage Notes

- If two servers are running on the same Windows machine, they will have to share the same $ORACLE_HOME, $REPORTS30_PATH, and $REPORTS30_TMP. However, each server listens to a different TNS name and has a unique configuration file, which can specify different sourcedir, cachedir, and tempdir settings, optionally on different drives.

- The minengine, maxengine, maxidle, and cachesize settings can be viewed and changed from the Reports Queue Manager. The queue administrator userID and password can also be changed from the queue manager. The configuration file can be overwritten by the server process when these settings change or when it needs to.

- Any optional arguments on the command line override the settings in the configuration file.

- Strings must be quoted if they contain spaces.

- Do not put spaces around the equal sign. (Follow the rules of a Report Builder command file.)

identifier

Is an internal setting that contains the encrypted queue administrator user ID and password. You should not attempt to modify it. If identifier is not specified or is deleted, or the configuration file is not present, anyone can supply any user ID and password from the Reports Queue Manager to log on as the queue administrator. Once someone has logged on in this way, the user ID and password they specified becomes the queue administrator user ID and password until changed from the Queue Manager.
maxconnect

Is the maximum number of processes that can communicate with the server process at any one time. This setting is the sum of the number of engines and clients, and must be greater than two (at least one engine and one client).

sourcedir

Is a path to be searched before $REPORTS30_PATH when searching for reports and other runtime files. This setting is useful when you have more than one Reports Server sharing the same $ORACLE_HOME because each Reports Server can search different directories.

cachedir

Is the cache for the Reports Server. cachedir can be set to any directory or logical drive on the machine. If it is not specified, the default is $ORACLE_HOME/report30/server/cache.

tempdir

Is a directory that will be used instead of $REPORTS30_TMP when creating temporary files. tempdir can be set to any directory or logical drive on the machine.

cachesize

Is the size of the cache in megabytes. If you expect to store the output of many of your reports in the Reports Server cache, you may want to increase this setting. If you do not expect to store a lot of output in the cache and have limited system resources, you may want to reduce it.

minengine

Is the minimum number of runtime engines the Reports Server should have available to run reports. The server process will attempt to keep at least this many engines active. Ensure that you have sufficient memory and resources available to accommodate this many engines.

maxengine

Is the maximum number of runtime engines available to the Reports Server to run reports. The server process will attempt to keep no more than this many engines active. Ensure that you have sufficient memory and resources available to accommodate this number of engines.

initengine

Is the initial number of runtime engines started by the Reports Server.
The server process will spawn this many engines when it is started. It will wait two minutes for these engines to connect to it and will shut itself down if they fail to do so. If the engines cannot connect in this amount of time, there is usually some setup problem.

**maxidle**

Is the maximum amount of time an engine is allowed to be idle before being shut down. Note that the Reports Server will not shut down the engine if doing so would reduce the number of available engines to less than minengine.

**security**

Is the security level (0, 1, 2, or 3) for accessing cached output files through the Reports Queue Manager. The default level of security is 1.

**Note:** This setting only controls access through the Queue Manager. Accessing output files via other methods (e.g., opening an output file in a browser or an editor) is controlled by whatever security you have in place for your file system. For example, if a job writes an ASCII output file to a particular directory, any user who has privileges on that directory will be able to look at the output in a text editor or viewer.

0 means that anyone can access a job’s cached output.

1 means that only a user whose user ID is identical to that of the user who ran the job can access the job’s cached output.

2 means that only the same process that sent the job can access the job’s cached output.

3 means that the cached output cannot be accessed.

**Note:** When someone uses the Reports Queue Manager to access output, the output is not encrypted when being moved across the network. In addition, with security level 1, anyone who has administrator privileges on any machine can fake the user ID to retrieve someone else’s output.

**englife**

Is the maximum number of reports that an engine will run before shutting itself down. The Reports Server will then bring up fresh engines for new requests.
2.2.4.1.6 Shutting Down the Reports Server

Windows NT service
1. Choose Start → Settings → Control Panel. Double-click Services, select the service, and click Stop.

Usage Notes
- Shutting down the Reports Server in this manner may result in temporary files not being removed automatically.

Windows 95
1. Double-click the Reports Server icon in the notification area of the taskbar. A dialog showing the status of the Reports Server is displayed.
2. Click Shutdown. This immediately shuts down the Reports Server without waiting for any pending reports to complete execution.

Usage Notes
- Shutting down the Reports Server in this manner may result in temporary files not being removed automatically.

Reports Queue Manager
1. Start the Reports Queue Manager (R30RQM32.EXE).
2. Choose Options → Privileges → Administrator.
3. Enter the queue administrator user ID and password and click OK. If you have not logged on as the queue administrator before, you can type in anything and that will become the queue administrator user ID and password until you change it.
4. Select Queue → Shutdown.
5. Select Normal or Immediate shutdown and click OK. Normal means wait until the queue is idle before shutting down. Immediate means shut down regardless of the state of the queue.

Usage Notes
- The Reports Queue Manager does not have to be running on the same machine as the Reports Server, i.e., you can administer the Reports Server remotely.
- If you use the Immediate shutdown option, temporary files may not be removed automatically.
2.2.4.2 Step 2: Setting up the Reports Web Cartridge or Web CGI

If you are using the Oracle Web Application Server, you can use the Reports Web Cartridge to handle job requests for the Reports Server. Otherwise, you can use the Reports Web CGI to handle job requests for the Reports Server. The online documents R30OWS.HTM and R30CGI.HTM also contain information about the Reports Web Cartridge and Web CGI.

2.2.4.2.1 Setting up the Reports Web Cartridge

Note: The following instructions are specific to Oracle Web Application Server 3.0. If you are using another Web server, refer to your Web server documentation for information about creating application cartridges.

1. Install Report Builder in its entirety or just the Reports Server, Web Cartridge, and Web CGI.

2. Install Oracle Web Application Server.
   
   Note: When you install the Web Application Server, use WEBSVR as the name of the default Web Listener (on Windows NT, this is the default). You specify the listener name in the Oracle Web Listener Configuration dialog.

3. Start the Admin listener. For example, type: wlctl30 start admin.

4. Start the WEBSVR listener. For example, type: wlctl30 start WEBSVR.

5. Start your browser.

6. Access the Oracle Web Application Server by navigating to the appropriate URL (http://<name of your machine or IP address>:<port number of Web Server Admin listener>).
   
   For example: http://myserver.com:9999.
   The Oracle Web Application Server appears.

7. Click Web Applications Server Manager to display the Administration page.

8. Click Oracle Web Application Server.

9. Click Cartridge Administration to display the Cartridge Administration page.

10. Click Add New Cartridge.
11. Click Add New Cartridge with Manual Configuration, then enter values for the following fields:

<table>
<thead>
<tr>
<th>In this field</th>
<th>Enter this value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cartridge Name</td>
<td>The cartridge name. For example: r30ows</td>
</tr>
</tbody>
</table>
| Object Path | The directory path and filename of the Reports Web Cartridge.  
  - On Microsoft Windows NT: \%ORACLE_HOME\bin\r30ows32.dll  
  - On UNIX: $ORACLE_HOME/r30owsm.so |
| Entry Point | r30ows_start  
r30ows_start is the name of the entry point routine in the cartridge shared library.  
**Note:** r30ows_start is the required value for Entry Point. |
| Minimum # of Instances | The minimum number of users that can simultaneously connect to the cartridge. |
| Maximum # of Instances | The maximum number of users that can simultaneously connect to the cartridge. |
| Virtual Path (under Virtual Paths) | At runtime, end users pass URLs (HTTP requests) to your Web server. If a request contains the virtual directory associated with your cartridge, the Web server automatically transfers the request to the Web Request Broker. The Web Request Broker in turn starts the cartridge associated with the virtual directory.  
When defining the Virtual Path, do not include a trailing slash. For example: /r30ows |
| Physical Path (under Virtual Paths) | The directory path for the Reports Web Cartridge:  
  - On Microsoft Windows NT: %ORACLE_HOME%\bin\  
  - On UNIX: $ORACLE_HOME/bin/ |

12. Click Register New Cartridge.

The Web Application Server registers your cartridge and creates a link for your cartridge in the Web Application Server Home page.
13. Click your cartridge link to display the Cartridge Configuration page.

14. Click **Cartridge specific parameters**, then enter the following parameters and parameter values:

   **Note:** Cartridge parameters and parameter values are case-sensitive.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Parameter Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>REPORTS30_WEBL</td>
<td>For example, on Windows NT:</td>
</tr>
<tr>
<td>OC</td>
<td>/cache/</td>
</tr>
<tr>
<td>REPORTS30_WEBL</td>
<td>For example, on Windows NT:</td>
</tr>
<tr>
<td>OC_TRANSLATED</td>
<td>c:\orant\mycache</td>
</tr>
</tbody>
</table>

15. (Optional) Set up optional cartridge parameters and environment variables to obtain further functionality from R30OWS.

16. Click **Modify Cartridge Configuration**.

17. To activate your changes, do the following:

   - Click **Listener**.
   - Click **Stop** to stop the WEBSVR listener.
   - Click **Start** to restart the WEBSVR listener.

2.2.4.2.2 Setting Up the Reports Web CGI for Windows 95/NT

1. Install Report Builder in its entirety or just the Reports Server, Web Cartridge, and Web CGI.

2. Copy the file `R30CGI32.EXE` (located in `$ORACLE_HOME\BIN`) into your Web server script directory.
3. Define the following in the system registry:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Parameter Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>REPORTS30_WEBLOC</td>
<td>For example, on Windows NT: /cache/</td>
</tr>
<tr>
<td>REPORTS30_WEBLOC_TRANSLATED</td>
<td>For example, on Windows NT: c:\orant\mycache</td>
</tr>
</tbody>
</table>

### 2.2.4.2.3 Setting Up the Reports Web CGI for UNIX

1. Install Report Builder in its entirety or just the Reports Server, Web Cartridge, and Web CGI.

2. Copy the file `r30cgim` (located in `$ORACLE_HOME/bin`) into your Web server script directory.

3. Set up a shell script in the Web Server script directory to define the environment variables listed below and invoke `r30cgim`.

**R30CGI Environment Variables:** `REPORTS30_WEBLOC`, `REPORTS30_WEBLOC_TRANSLATED`

**Other Required Environment Variables:** `DISPLAY`, `ORACLE_HOME`, `PATH`, `ORACLE_SID`, `TNS_ADMIN`, `TWO_TASK`, `REPORTS30_PATH`, `LD_LIBRARY_PATH`

**Example script:**

```bash
#!/bin/sh

# cgi-specific variables
REPORTS30_WEBLOC=myvirtualdir
REPORTS30_WEBLOC_TRANSLATED=/blah/wwwroot/myactualdir

# Other required variables
DISPLAY=mymachine:0.0; export DISPLAY
ORACLE_HOME=/blah/myoracledir; export ORACLE_HOME
PATH=.:${ORACLE_HOME}/bin; export PATH
TNS_ADMIN=${ORACLE_HOME}/network/admin; export TNS_ADMIN
TWO_TASK=mytwotask; export TWO_TASK
LD_LIBRARY_PATH=${ORACLE_HOME}/lib; export LD_LIBRARY_PATH
```
2.2 Setting Up the Developer/2000 Server

# Here's the call to the cgi executable
./r30cgim

2.2.4.2 Cartridge Parameters and Environment Variables
Parameters and environment variables are used to further configure the Reports Web Cartridge and Web CGI.

- Section 2.2.4.2.5, “ORACLE_HOME”
- Section 2.2.4.2.6, “LD_LIBRARY_PATH”
- Section 2.2.4.2.7, “REPORTS30_WEBLOC”
- Section 2.2.4.2.8, “REPORTS30_WEBLOC_TRANSLATED”
- Section 2.2.4.2.9, “REPORTS30_OWSHELP”
- Section 2.2.4.2.10, “REPORTS30_OWSMAP”
- Section 2.2.4.2.11, “REPORTS30_OWSDIAGBODYTAGS”
- Section 2.2.4.2.12, “REPORTS30_OWSDIAGHEADTAGS”
- Section 2.2.4.2.13, “REPORTS30_OWSPATHONLYURL”
- Section 2.2.4.2.14, “REPORTS30_OWSNODIAG”
- Section 2.2.4.2.15, “REPORTS30_CGIHELP”
- Section 2.2.4.2.16, “REPORTS30_CGIMAP”
- Section 2.2.4.2.17, “REPORTS30_CGIDIAGBODYTAGS”
- Section 2.2.4.2.18, “REPORTS30_CGIDIAGHEADTAGS”
- Section 2.2.4.2.19, “REPORTS30_CGINODIAG”
- Section 2.2.4.2.20, “REPORTS30_CGIPATHONLYURL”
- Section 2.2.4.2.21, “USER_NLS_LANG or NLS_LANG”

Note: The way these parameters/variables are set varies. On Windows 95/NT, they are set in the System Registry. On UNIX, they are set with the `setenv` command. For the Reports Web Cartridge, they are set as parameters (i.e., in the Oracle Web Application Server).

2.2.4.2.5 ORACLE_HOME
Specifies the file directory where all Oracle software is installed (for example, `c:\orant\` on Windows NT). The location of Oracle shared libraries, NLS resource and message files is derived from this variable. On Windows, it is usually set up during the installation process.
2.2.4.2.6 **LD_LIBRARY_PATH**  On UNIX, specifies directory path to search for shared libraries.

2.2.4.2.7 **REPORTS30_WEBLOC**  Specifies where the Web server looks for the Reports Server output (file cache). It is a virtual directory, as defined in the Web server configuration file. It may be mapped directly to the Reports Server file cache directory (defined in the .ora file for the Reports Server being used) only if that directory is mapped from the Web server machine. REPORTS30_WEBLOC is usually mapped to the physical directory defined by the REPORTS30_WEBLOC_TRANSLATED parameter, which is valid regardless of whether the Reports Server and the Web server share the same file system.

If this parameter is not set, R30OWS or R30CGI will not display HTML/PDF output in the client browser ("Cannot access report output" message will result), although the report will run and output will be produced.

**Examples:**

**Note:** cachedir is set in the Reports Server configuration file. REPORTS30_WEBLOC and REPORTS30_WEBLOC_TRANSLATED are set as parameters (i.e., in the Oracle Web Application Server) for the Reports Web Cartridge and as environment variables for the Reports Web CGI.

```bash
cachedir="c:\orant\report30\server\cache"
reports30_webloc="/"
reports30_webloc_translated="C:\orant\ows21\new"
```

The cache directory of the Reports Server is set to the default value. The REPORTS30_WEBLOC variable means that a Web listener is using "/" as a virtual directory path. The REPORTS30_WEBLOC_TRANSLATED variable sets the physical directory name of the web listener's default virtual root directory.

```bash
cachedir="C:\servercache"
reports30_webloc="/repout"
reports30_webloc_translated="C:\repdir"
```

This case is similar to the first example, except that the settings are not the default directories.

```bash
cachedir="C:\orant\ows21\new"
reports30_webloc="/"
```

In this case, the Web listener and the Reports Server cache share the same directory. You should not set REPORTS30_WEBLOC_TRANSLATED in this situation because it will cause problems in retrieving HTML output.
2.2.4.2.8 REPORTS30_WEBLOC_TRANSLATED Specifies which directory the report output will be transferred to on the Web server machine if the Reports Server and Web server do not share file systems. This must be the absolute physical path of the virtual directory defined in REPORTS30_WEBLOC in order for there to be no conflict. If the value of this variable would be the same as the cachedir setting in the Reports Server configuration file, you should not specify a value for it.

If this variable is set to an invalid value, R30OWS or R30CGI will not be able to display HTML/PDF output in the client browser ("Cannot access report output" message will result), although the report will run and output will be produced.

See the examples in Section 2.2.4.2.7, “REPORTS30_WEBLOC”.

2.2.4.2.9 REPORTS30_OWSHELP For the Reports Web Cartridge, defines URL/URI of the R30OWS help file, which will be navigated to when R30OWS is invoked with the empty request:

http://your_webserver/r30ows?

e.g., setting it to http://www.yahoo.com will go to that URL; setting it to myhelpfile.htm will display the file:

http://your_webserver/myhelpfile.htm

If this parameter is not defined, a default help screen will be displayed.

2.2.4.2.10 REPORTS30_OWSMAP For the Reports Web Cartridge, defines fully qualified filename/location of the R30OWS map file (if map file configuration is used), e.g., C:\ORANT\REPORT30\MYMAP.DAT.

2.2.4.2.11 REPORTS30_OWSDIAGBODTAGS For the Reports Web Cartridge, specifies HTML tags which will be inserted as a <BODY…> tag in the R30OWS diagnostic/debugging output. For instance, you may want to use this environment to set up text/background color, image, etc.

2.2.4.2.12 REPORTS30_OWSDIAGHEADTAGS For the Reports Web Cartridge, specifies HTML tags which will be inserted between <HEAD> …</HEAD> tags in the R30OWS diagnostic/debugging output. For instance, you may want to use this environment to set up <TITLE> or <META…> tags, etc.
2.2.4.2.13 REPORTS30_OWSPATHONLYURL For the Reports Web Cartridge, when defined (the value does not matter), it causes R30OWS to use path-only URLs (e.g., "/rscache/..." instead of "http://mywebsrv.com:1550/rscache/...") in HTTP redirection to report output and in the R30OWS-generated diagnostic output. This is useful to force the client browser to retain the original URL request in its "Location:" field after retrieving the report output. Normally, R30OWS will switch the URL to the location of the report output (i.e., the directory specified by REPORTS30_WEBLOC), and will show the name of the report output file which, just having been generated, has a rather uninformative name such as 12345679.htm. By setting this parameter, this "URL switching" can be avoided. This will work only if all URLs are using the default port (typically 80).

The drawback to setting this parameter, however, is that any subsidiary files (such as image files) dependent on the main report output file will not be retrieved by the browser (although they are created successfully in the pointed to by REPORTS30_WEBLOC). This can be worked around either by specifying a <BASE HREF="..."> or <META HREF="..."> tag (with the main file location as a full or partial URL) in the report definition so that it is included in the main report output file, or by setting REPORTS30_WEBLOC to point to the location of the R30OWS executable.

Setting REPORTS30_OWSPATHONLYURL can also be used to workaround certain Intranet/Internet configurations where the same machine may have several different IP addresses simultaneously (with several network cards), because it removes the dependence of the HTTP redirection on the machine name or IP address.

2.2.4.2.14 REPORTS30_OWSNODIAG For the Reports Web Cartridge, when defined, disables all debugging/diagnostic output from R30OWS:

http://your_webserver/r30ows/help?

will not work when REPORTS30_OWSNODIAG is defined.

2.2.4.2.15 REPORTS30_CGIHLP For the Reports Web CGI, defines URL/URI of the R30CGI help file, which will be navigated to when R30CGI is invoked with the empty request:

http://your_webserver/your_virtual_cgi_dir/r30cgi32.exe?

For example, setting REPORTS30_CGIHLP to http://www.yahoo.com will go to that URL; setting it to myhelpfile.htm will display the file:

http://your_webserver/myhelpfile.htm

If this variable is not defined, a default help screen will be displayed.
2.2.4.2.16 REPORTS30_CGIMAP For the Reports Web CGI, defines fully qualified filename/location of the R30CGI map file if map file configuration is used. For example:

c:\orant\report30\mymap.dat

2.2.4.2.17 REPORTS30_CGIDIA GBODYTAGS For the Reports Web CGI, specifies the HTML tags that will be inserted as <BODY...> tags in the R30CGI diagnostic/debugging output. For example, you can use this environment variable to set up text/background color and image.

2.2.4.2.18 REPORTS30_CGIDIAGHEADTAGS For the Reports Web CGI, specifies HTML tags which will be inserted between <HEAD> ... </HEAD> tags in the R30CGI diagnostic/debugging output. For example, you may want to use this environment to set up <TITLE> or <META...> tags.

2.2.4.2.19 REPORTS30_CGINODIAG For the Reports Web CGI, when defined, disables all debugging/diagnostic output from R30CGI:

http://your_webserver/cgi-bin/r30cgi32.exe/help?

will not work when REPORTS30_CGINODIAG is defined.

2.2.4.2.20 REPORTS30_CGIPATHONLYURL For the Reports Web CGI, when defined (the value does not matter), forces R30CGI to return any URLs to the browser as partial, path-only URLs; i.e., instead of returning:

Location: http://your_webserver:port/path

it will return only:

Location: /path

This variable can be used to resolve problems with Web server setups that use multiple ports, including a default port.

2.2.4.2.21 USER_NLS_LANG or NLS_LANG Defines NLS Language used in your URL request as well as HTML/PDF output (using Oracle NLSRTL Language IDs). For example:

American_America.WE8ISO8859P1
2.2.4.3 Step 3: Specifying URL run requests

When specifying URL run requests to the Reports Web Cartridge or CGI, you should:

- Make sure all report filenames and connection strings are valid on the Reports Server’s file system.
- Specify at least three parameters: SERVER, REPORT, and USERID.
- If you are using the Reports Web CGI on Windows, use R30CG132.EXE (case insensitive). On UNIX, use r30cgim (case sensitive) unless you use a shell script to invoke CGI runtime, in which case you should use the name of the script that calls r30cgim.

The examples below are equivalent to the following command line:

```
R30CLI SERVER=your_repserver REPORT=breakb.rdf USERID=scott/tiger@MYDB
DESTYPE=CACHE DESFORMAT=HTML
```

The online documents R30OWS.HTM and R30CGI.HTM also contain information about URL run requests. The online document R30MTSUS.HTM and Reports Runtime help system contain a complete description of the command line parameters and semantics.

2.2.4.3.1 A non-URL-encoded (non-standard) request:

**Web Cartridge:**

```
http://your_webserver/r30ows?server=your_repserver+report=breakb.rdf+userid=scott/tiger@MYDB+destype=cache+desformat=HTML
```

**CGI:**

```
http://your_webserver/your_virtual_cgi_dir/r30cgi32.exe?server=your_repserver+report=breakb.rdf+userid=scott/tiger@MYDB+destype=cache+desformat=HTML
```

2.2.4.3.2 A URL-encoded standard request:

**Web Cartridge:**

```
http://your_webserver/r30ows?server%3Dyour_repserver%2Creport%3Dbreakb%2Erdf%2Cuserid%3Dscott%2Ftiger%40MYDB%2Cdestype%3Dcache%2Cdesformat%3DHTML
```
2.2 Setting Up the Developer/2000 Server

CGI:
http://your_webserver/your_virtual_cgi_dir/r30cgi32.exe?server%3Dyour_repserver+report%3Dbreakb%2Erdf+userid%3Dscott%2Ftiger%40MYDB+destype%3Dcache+desformat%3DHTML

2.2.4.3 A non-URL-encoded (non-standard) form-style request:

Web Cartridge:
http://your_webserver/r30ows?server=your_repserver&report=breakb.rdf&userid=scott/tiger@MYDB&destype=cache&desformat=HTML

CGI:
http://your_webserver/your_virtual_cgi_dir/r30cgi32.exe?
server=your_repserver&report=breakb.rdf&userid=scott/tiger@MYDB&destype=cache&desformat=HTML

2.2.4.3.4 A URL-encoded standard form-style request:

Web Cartridge:
http://your_webserver/r30ows?server=your_repserver&report=breakb%2Erdf&userid=scott%2Ftiger%40MYDB&destype=cache&desformat=HTML

CGI:
http://your_webserver/your_virtual_cgi_dir/r30cgi32.exe?
server=your_repserver&report=breakb%2Erdf&userid=scott%2Ftiger%40MYDB&destype=cache&desformat=HTML
2.2.4.3.5 An HTML form:

Web Cartridge:

```html
<!DOCTYPE HTML>
<!--Form Action is R30OWS URL-->
<!--Form METHOD=POST ACTION="http://your_webserver/r30ows?"-->
<!--Parameters not exposed to user are hidden-->
<INPUT name=foo type=hidden value=bar>
<CENTER><H1>Set Reports Multi-tier Server Parameters: </H1>
<br/>
Report Name:<INPUT name=report type=text value="breakb.rdf"><br/>
Database Connection:<INPUT name=userid type=text value="scott/tiger@mydb"><br/>
Output Format:<SELECT name=desformat>
  <OPTION value=HTML selected> HTML
  <OPTION value=PDF> PDF
</SELECT> <br/>
<br/><INPUT type=submit value="Run Report!"> 
</CENTER> </FORM> </HTML>
```

CGI:

```html
<!DOCTYPE HTML>
<!--Form Action is R30CGI URL-->
<!--Form METHOD=POST ACTION="http://your_webserver/your_virtual_cgi_dir/r30cgi32.exe?"-->
<!--Parameters not exposed to user are hidden-->
<INPUT name=foo type=hidden value=bar>
<CENTER><H1>Set Reports Multi-tier Server Parameters </H1>
<br/> Report Name: <INPUT name=report type=text value="breakb.rdf"><BR>
Database Connection: <INPUT name=userid type=text value="scott/tiger@mydb"><BR>
Output Format: <SELECT name=desformat> <OPTION value=HTML selected> HTML <OPTION value=PDF> PDF </SELECT> <BR>
<br/> <INPUT type=submit value="Run Report!"> </CENTER> </FORM> </HTML>
```
2.2 Setting Up the Developer/2000 Server

2.2.4 Step 4 (Optional): Using the URL mapping feature

Both the Reports Web Cartridge and Web CGI implement URL mapping. When URL mapping is enabled, the first argument in the URL is treated as a special key. This key maps to command line parameter-value combinations in a key mapping file set up by a Web site administrator.

URL mapping is useful for:

- shortening the URL, making it more convenient to use
- remapping the URL run configuration without having to change the original URL
- standardizing several typical run configurations for the organization
- hiding certain parameters from the end user (e.g., database connect string)
- restricting end user choice as to which parameters may be used for the request

2.2.4.4.1 Enabling URL Mapping

URL Mapping is enabled when either of the two conditions below are met:

- The cartridge parameter (REPORTS30_OWSMAP) or environment variable (REPORTS30_CGIMAP) on the Web server machine specifies the name of a valid key mapping file.
- A valid file with the standard filename (OWSCMD.DAT or CGICMD.DAT) is present in $ORACLE_HOME/REPORT30.

Usage Notes

- When URL Mapping is enabled, all R30OWS or R30CGI URLs will be treated as if the first argument was a key. The key mapping file will be searched for this key. If the key is found, its defined value will be substituted into the command line for the Reports Server. If it is not found, an error is generated.
- When submitting a URL via an HTML form, the key is coded as an input of type hidden.
- Nested mappings are not supported.

2.2.4.4.2 Key Mapping File Syntax

Each key entry is defined as key:value where:

- key is the first argument of the URL request (case insensitive).
value is a list of command line parameter-values, special parameters, or a combination of the two.

Special parameters are represented as %X, where X identifies the parameter. Currently recognized special parameters are:

%\n : argument number \n from the original URL request, where \n is an integer from 0 - 9. Note that while %0 is a valid value it actually represents the key itself and, as such, is unnecessary in the key mapping file.

%* : entire contents (all arguments) of original URL request.

%P : request for a parameter form in HTML format. To ensure the parameter form only appears once, it generates the PARAMFORM=HTML construction on the first submission of the URL and PARAMFORM=NO on the final parameter form submission.

%PT : request for a parameter form in HTMLTABLE format. To ensure the parameter form only appears once, it generates the PARAMFORM=HTMLTABLE construction on the first submission of the URL and PARAMFORM=NO on the final parameter form submission.

%PC : request for a parameter form in HTMLCSS format. It generates the PARAMFORM=HTMLCSS construction on the first submission of the URL and PARAMFORM=NO on the final parameter form submission.

Syntax Rules

■ There should be no space between the key and the semicolon.
■ Extra spaces are ignored after the semicolon.
■ Multi-line entries are allowed.
■ There must be at least one blank line between two successive keys.
■ Any line commencing with ; (a semicolon) is treated as a comment and ignored.
■ Comments inside a key or value are not allowed; i.e., comments can be used only before or after a key definition.
■ NLS language is supported and can be used (encoding should match the one used in HTML request - no language conversion of any kind is attempted).
■ The HTML format is mapped to the HTMLTABLE format by default.
Example OWSCMD.DAT file

; Here is a basic key which will take all the arguments
; supplied in the rest of the URL:
key1: %*
; This key does the same thing for a self-made HTML
; form from which the report is to be
; submitted. In the form, you might have
; <INPUT NAME=key2 TYPE=hidden VALUE = "">
; as the first control, which would translate to "run=
; on the resulting command line, so you need a key with
; that name.
key2=: %*
; The next key enables you to hide the connect string from
; the user, and takes the other arguments from the rest of
; the URL. Note that if the rest of the URL also contains a
; connect string, an error will result.
key3: userid=scott/tiger@mydb %*
; This one supplies the report name, takes the first two
; arguments from the rest of the URL, and ignores subsequent
; parameters:
key4: report=your_report.rdf %1 %2
; Finally, here's a key that encapsulates all
; parameters required for the report to run.
key5: server=your_repserver userid=scott/tiger@mydb report=your_report.rdf
desformat=html

Example CGICMD.DAT file

; Here is a basic key which will take all the arguments
; supplied in the rest of the URL:
key1: %*
; This key does the same thing for a self-made HTML form
; from which the report is to be submitted. In the form, you
; might have <INPUT NAME=key2 TYPE=hidden VALUE = "">
; as the first control, which would translate to "run=" on
; the resulting command line, so you need a key with that name.
key2=: %*
; The next key enables you to hide the connect string from
; the user, and takes the other arguments from the rest of
; the URL. Note that if the rest of the URL also contains a
; connect string, an error will result.
key3: userid=scott/tiger@mydb %*
; This one supplies the report name, takes the first two
; arguments from the rest of the URL, and ignores subsequent
; parameters:
key4: report=your_report.rdf %1 %2
; Finally, here's a key that encapsulates
; all parameters required for the report to run.
key5: server=your_repserver userid=scott/tiger@mydb report=your_report.rdf
desformat=html

### 2.2.4.5 Step 5: Deploying reports on the Web

You create the Reports Server in the same way you do for any other environment, using the Report Wizard and the Report Editor. You can then add Web functionality to the report using the Web Wizard and the Report Editor. The Web Wizard enables you to quickly add basic Web functionality (e.g., bookmarks). The Report Editor enables you to add more advanced functionality (e.g., custom Report Escapes and URLs).

After the reports you want to deploy on the Web are ready, copy the .RDF files to the appropriate directory on the application server file system. Be sure to generate the .RDF files on the same platform as the application server on which you will deploy them.

You can deploy .RDF files in any directory on your application server, provided you have defined the cartridge parameters and environment variables correctly. Refer to Section 2.2.4.2.4, “Cartridge Parameters and Environment Variables”.

Once you have copied .RDF files to the application server, you must link them to a Web page and notify end users of their availability. For more information, refer to Section 2.2.1.3.6, “Step 3c: Providing URLs of your Web-enabled applications”.
## 2.2.5 Troubleshooting your Reports Server configuration

<table>
<thead>
<tr>
<th>Problem</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reports Server appears to hang when you start it.</td>
<td>You may have made a syntactical error in the tnsnames.ora file and the Reports Server cannot resolve the tnsname.</td>
</tr>
<tr>
<td>You get the error “Daemon failed to listen to port.”</td>
<td>If you start up a Reports Server that is listening to the same port as an already running Reports Server, you will get this error.</td>
</tr>
<tr>
<td>You get an error about being unable to initialize the printer (REP-3002)</td>
<td>Make sure that the Reports Server has access to printers. On Windows NT, the System Account does not usually have access to printers. It could be that you installed the Reports Server as an NT service and used the System Account or another account without printer access in the Log On As field. You must specify an account in the Log On As field that has printer access.</td>
</tr>
</tbody>
</table>
| Upon starting the Reports Server, you get server specific error 186. | Typically this indicates a problem in tnsnames.ora or sqlnet.ora. Check the entry for the Reports Server in tnsnames.ora. A typical entry should look something like the following:  
repserver.world = (ADDRESS=(PROTOCOL=tcp)(HOST=144.25.87.182)(PORT=1951))  
If your tnsnames.ora file appears to be correct, check your sqlnet.ora file. Good default settings to use in this file are:  
TRACE_LEVEL_CLIENT=OFF  
names.directory_path = (TNSNAMES)  
names.default_domain = world  
names.default_zone = world  
Lastly, be sure that your installed version of SQL*Net is not older than the version that came with the Reports Server. |
| Error reported when opening the report. | Check the name and extension carefully. On UNIX machines, the actual report name must be in the same case as specified in the URL. If you are using Windows Explorer in Windows, be sure not to hide extensions for the displayed files that you are copying and renaming. (Check View → Options in the Explorer window.) This prevents you from creating files with names like “your_report.rdf.txt.” Alternatively, use a DOS window for file manipulation. |

### Table 2-1 Troubleshooting the Reports Server
Problems running the Reports Server as a Windows NT Service.

If you install the Reports Server service to run under a user other than SYSTEM, make sure that the user account:

- Has the Password Never Expires option selected in the User Manager.
- Has membership in the appropriate groups to run the Reports Server and access the report files.
- Has at least print permission to a default printer.
- Can log on to a service. Choose Start → Programs → Administrative Tools → User Manager, then Policies → User Rights. Check Show Advanced User Rights. From the Right list, choose Log on as a service. If the user is not already in the Grant To list, click the Add button.

When starting the service, you may need to explicitly specify the domain as well as the username (username/domain).

If you get a Windows NT error reporting that the service failed and returning the error message number, you can look up the message number in the Report Builder online help.

ops$ account is not working.

ops$ accounts are not supported by the Reports Server. If you pass a command line with USERID=/ to the Reports Server, an error is generated because it will try to use the username of the Reports Server process rather than the username of the client.

Database roles not working as expected.

If you are using database roles, the Reports Server gets and then sets the default roles for the job request’s database connection. If the default roles require a password, the Reports Server will log off and then back on to the database. As a result, it is best to include roles that require passwords in the report itself using the Role Name report property.

Since the Reports Server gets and then sets the default roles on a per job basis, you cannot share roles between jobs. This is done to preserve security.

Table 2-1 Troubleshooting the Reports Server
#### Problem
Reports are not running.

#### Solution
Make sure the Web server is responding. Type:

```
http://your_webserver
```

in your browser URL window. A Web server administration page should appear; if not, see your Web server installation documentation.

Make sure your Web Cartridge or CGI executable has been found and is responding. For Windows 95/NT, type one of the following:

```
http://your_webserver/your_virtual_cgi_dir/r30cgi32.exe
http://your_webserver/r30ows
```

For UNIX, type:

```
http://your_webserver/your_virtual_cgi_dir/r30cgim
```

in your browser URL field.

A help page should appear. If it does not, check the of your `virtual_bin_directory` (usually called `cgi-bin`) in your Web server configuration file. It should be mapped to an existing physical directory on your Web server. You must have a copy of the R30CGI executable in this physical directory.

Make sure that the environment variable `REPORTS30_OWSNODIAG` or `REPORTS30_CGINODIAG` is not defined, otherwise all diagnostic output will be disabled. Test this by typing one of the following:

```
http://your_webserver/your_virtual_cgi_dir/r30cgi32.exe/showenv?
http://your_webserver/r30ows/showenv?
```

This will also allow you to view the other cartridge parameters or environment variables.

Check the list of cartridge parameters or environment variables and make sure that `REPORTS30_WEBLOC` and `REPORTS30_WEBLOC_TRANSLATED` both are defined. The former is defined to be a virtual directory on the Web server; the latter is defined to be the absolute path of the directory which the virtual directory maps to.

<table>
<thead>
<tr>
<th>Table 2-1 Troubleshooting the Reports Server</th>
</tr>
</thead>
</table>
Reports are not running. Make sure the environment variable REPORTS30_PATH is defined.

If you do not plan to use URL mapping, make sure that REPORTS30_OWSMAP or REPORTS30_CGIMAP is undefined.
Check to be sure that there is no file named owscmd.dat or cgicmd.dat in the ORACLE_HOME\REPORT30\ directory.
Confirm that a map file is not found by typing one of the following:
http://your_webserver/you_virtual_cgi_dir/r30cgi32.exe/showmap?
http://your_webserver/r30ows/showmap?

Try running a simple report to your browser, by typing one of the following:
http://your_webserver/your_virtual_cgi_dir/r30cgi32.exe?report=your_report.rdf+userid=scott/tiger@mydb+desformat=html
http://your_webserver/r30ows?server=your_repserver+report=your_report.rdf+userid=scott/tiger@mydb+desformat=html

If the report does not display, check to be sure that:
- Your_report.rdf runs correctly from Report Builder or Reports Runtime
- Your_report.rdf is located in a directory specified under REPORTS30_PATH.
- The database connection string is correct.

Remember that the Reports Server must have access to the report and any external files used by the report.

When sending a report to the Reports Server, you should only use the In Report value for parameters if they have their values explicitly set in the report definition. For example, suppose that you are launching a report from the Reports Queue Manager (Job→New). If you specify In Report for the Report Mode and Orientation parameters and neither of them has a value specified in the report definition, the job will fail.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reports are not running.</td>
<td>Make sure the environment variable REPORTS30_PATH is defined. If you do not plan to use URL mapping, make sure that REPORTS30_OWSMAP or REPORTS30_CGIMAP is undefined. Check to be sure that there is no file named owscmd.dat or cgicmd.dat in the ORACLE_HOME\REPORT30\ directory. Confirm that a map file is not found by typing one of the following: http://your_webserver/you_virtual_cgi_dir/r30cgi32.exe/showmap? http://your_webserver/r30ows/showmap? Try running a simple report to your browser, by typing one of the following: http://your_webserver/your_virtual_cgi_dir/r30cgi32.exe?report=your_report.rdf+userid=scott/tiger@mydb+desformat=html http://your_webserver/r30ows?server=your_repserver+report=your_report.rdf+userid=scott/tiger@mydb+desformat=html If the report does not display, check to be sure that: Your_report.rdf runs correctly from Report Builder or Reports Runtime Your_report.rdf is located in a directory specified under REPORTS30_PATH. The database connection string is correct. Remember that the Reports Server must have access to the report and any external files used by the report. When sending a report to the Reports Server, you should only use the In Report value for parameters if they have their values explicitly set in the report definition. For example, suppose that you are launching a report from the Reports Queue Manager (Job→New). If you specify In Report for the Report Mode and Orientation parameters and neither of them has a value specified in the report definition, the job will fail.</td>
</tr>
</tbody>
</table>

Table 2-1 Troubleshooting the Reports Server
PDF reports run, but HTML reports generate an error saying that the requested URL was not found. It could be that REPORTS30_WEBLOC_TRANSLATED points to the same location as the Reports Server cache (i.e., cachedir in the Reports Server configuration file). In this scenario, the Reports Web CGI is trying to copy the report output from the cache directory to the root directory of the Web listener, but, since those directories are the same, you receive an error. Check your REPORTS30_WEBLOC_TRANSLATED environment variable. If cachedir is the same as your REPORTS30_WEBLOC_TRANSLATED, delete the value of REPORTS30_WEBLOC_TRANSLATED.

If cachedir is different than REPORTS30_WEBLOC_TRANSLATED, change the value of REPORTS30_WEBLOC_TRANSLATED to the absolute physical path of the virtual directory defined in REPORTS30_WEBLOC (e.g., if REPORTS30_WEBLOC=/repout and you have /repout specified in your virtual directory mappings in your Web listener, then REPORTS30_WEBLOC_TRANSLATED needs to be set to the physical directory corresponding to /repout as specified in your Web listener’s directory mapping configuration).

URL mapping is not working. Make sure you have a valid key mapping file. It must either be named owscmd.dat or cgicmd.dat in the REPORT30 directory, or named according to the value of the environment variable REPORTS30_OWSMAP or REPORTS30_CGIMAP.

Test running a report by typing one of the following:

- http://your_webserver/your_virtual_cgi_dir/r30cgi32.exe?yourkey+report=your_report.rdf+userid=scott/tiger@mydb+desformat=html
- http://your_webserver/r30ows?yourkey+server=your_repserver+report=your_report.rdf+userid=scott/tiger@mydb+desformat=html

where your key is a valid key entry in the key mapping file.

Cannot shutdown the queue from the Reports Queue Manager. You should not leave username and password blank the first time that you log in as the administrator. The first time that you log in as the queue administrator from the Reports Queue Manager (Options→Privileges→Administrator), you can specify any username and password. The username and password that you specify the first time will be the administrator’s until you change it.

**Table 2-1 Troubleshooting the Reports Server**
2.2.6 Configuring the Graphics Server

To deploy Graphics Builder displays on the Web, refer to the following instructions:

- Section 2.2.6.1, “Creating virtual directories on your Web server”
- Section 2.2.6.2, “Creating the Graphics Server cartridge”
- Section 2.2.6.3, “Making the Graphics Builder web cartridge available to end users”

2.2.6.1 Creating virtual directories on your Web server

- To create pointers to various physical directories on your application server, create virtual directories on your Web server.

For more information about creating virtual directories, refer to your Web Server documentation.

2.2.6.2 Creating the Graphics Server cartridge

Note: The following instructions are specific to Oracle Web Application Server 3.0. If you are using another Web server, refer to your Web server documentation for information about creating application cartridges.

1. Install Oracle Web Application Server, if you haven’t already.

   Note: When you install the Web Application Server, use WEBSVR as the name of the default Web Listener (on Windows NT, this is the default). You specify the listener name in the Oracle Web Listener Configuration dialog.

2. (UNIX only) Set the GRAPHICS_WEB_DIR and OWS_IMG_DIR environment variables.
   - Set GRAPHICS_WEB_DIR to the full directory path that contains the .OGDs that you want to run on the Web.
   - Set OWS_IMG_DIR to the location of the /ows-img/ virtual directory. You define the /ows-img/ virtual directory in the Oracle Web Application Server Administration page.

   Note: On UNIX, you must start the listener from the shell where you set the GRAPHICS_WEB_DIR and OWS_IMG_DIR environment variables.

3. Start the Admin listener. For example, type: wlc130 start admin.

4. Start the WEBSVR listener. For example, type: wlc130 start WEBSVR.

5. Start your browser.
6. Access the Oracle Web Application Server by navigating to the appropriate URL (http://<name of your machine or IP address>:<port number of Web Server Admin listener>).

   For example: http://myserver.com:9999.

   The Oracle Web Application Server appears.

7. Click **Web Applications Server Manager** to display the Administration page.

8. Click **Oracle Web Application Server**.

9. Click **Cartridge Administration** to display the Cartridge Administration page.

10. Click **Add New Cartridge**.

11. Click **Add New Cartridge with Manual Configuration**, then enter values for the following fields:

<table>
<thead>
<tr>
<th>In this field...</th>
<th>Enter this value...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cartridge Name</td>
<td>The cartridge name. For example: OGWEB</td>
</tr>
<tr>
<td>Object Path</td>
<td>The directory path and filename of the Forms Cartridge Handler.</td>
</tr>
<tr>
<td></td>
<td>On Microsoft Windows NT: %ORACLE_HOME%/bin/gw30w32.dll</td>
</tr>
<tr>
<td></td>
<td>On UNIX: $ORACLE_HOME/lib/libgw30.so</td>
</tr>
<tr>
<td>Entry Point</td>
<td>GWWRBMain</td>
</tr>
<tr>
<td></td>
<td>GWWRBMain is the name of the entry point routine in the cartridge shared library.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong>: GWWRBMain is the required value for Entry Point.</td>
</tr>
<tr>
<td>Minimum # of Instances</td>
<td>The minimum number of users that can simultaneously connect to the cartridge. For example: 0</td>
</tr>
<tr>
<td>Maximum # of Instances</td>
<td>The maximum number of users that can simultaneously connect to the cartridge. For example: 100</td>
</tr>
</tbody>
</table>
12. Click Register New Cartridge.

The Web Application Server registers your cartridge and creates a link for your cartridge in the Web Application Server Home page.

13. Click your cartridge link to display the Cartridge Configuration page.

14. Click Cartridge specific parameters, then enter the following parameters and parameter values:

   **Note:** Cartridge parameters and parameter values are case-sensitive.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Parameter Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>GW_TIMEOUT</td>
<td>The number indicating the time interval (in minutes) for which the display will remain open. When the time interval specified here has elapsed, the display is automatically closed. For example:</td>
</tr>
</tbody>
</table>
15. (Optional) Enter the following optional cartridge parameters and parameter values.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Parameter Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>GW_IMAGES_USE_FILES</td>
<td>The value of this parameter determines whether the Graphics Server creates temporary GIF image files on the server (in the web_tmp directory in the img directory in the web server path) when returning HTML pages that show Graphics displays. If the value is NO, then the HTML page instead references the Graphics Server (a second time) to retrieve the image data, which is &quot;streamed&quot; back to the client by the server as binary content without creating a temporary file on the server. The default value of this parameter (if it is omitted entirely) is YES. Note: Creating temporary GIF files can introduce an administrative task of periodically removing these files from the server (once you are certain they are not being used). However, the Graphics cartridge automatically removes old image files. Using temporary files can result in some &quot;unused&quot; image files on the server; however, this is offset by a small performance benefit, since using &quot;streamed&quot; images is slightly slower. Using &quot;streamed&quot; images is slower because image requests must also be routed through the Web Request Broker to the Graphics Server (instead of ordinary requests to the server for image files which are optimized operations).</td>
</tr>
</tbody>
</table>
16. Click **Modify Cartridge Configuration**.

17. To activate your changes, do the following:
   - Click **Listener**.
   - Click **Stop** to stop the WEBSVR listener.
   - Click Start to restart the WEBSVR listener.

18. (UNIX only) Before running the cartridge, create the directory 
    `$OWS_IMG_DIR/web_tmp`, then give it global write permissions.

### GW_LINKS_CLOSE_PREV

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Parameter Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>GW_LINKS_CLOSE_PREV</td>
<td>Determines whether the Graphics Server closes the current OGD file when navigating to another OGD file via an embedded hyperlink (i.e., URL).</td>
</tr>
</tbody>
</table>

If the parameter value is **YES**, and an embedded URL is used to open another OGD on any Graphics Server (i.e., if it contains the string "openfile"), then the current OGD will be closed and the new one opened.

This conserves resources on the server, since the user no longer needs the first OGD to remain open. **Note:** Other types of URLs (e.g., to another web-page or HTML object) will operate as usual, and the OGD containing these links will remain open after executing them, regardless of this parameter value.

If the user returns to the first OGD, either via the "back" button on their browser or via another URL, they will see a "fresh" version of this document; any changes they made before will not appear.

The default value of this parameter is **YES**. This parameter value may be overridden by setting the bind-variable **OG_URL_CLOSE_THIS_DOC** to **YES** or **NO** in the OGD PL/SQL function that invokes the embedded URL (i.e., via **OG_URL** and **OG_URL_TARGET**). **Note:** The value of **OG_URL_CLOSE_THIS_DOC** must be explicitly set each time (before a link is executed) to override this parameter value, because it is only meant for special cases.

### GW_WRITE_TRACE_FILE

The value of this parameter determines whether the Graphics Server writes a "trace" file to the server's filesystem. The trace file can be used by an administrator for debugging purposes.

The name of this file is `gwtrace.txt`.

Because trace files are required in special situations, the default value of this parameter is **NO**.
For example, type:

```bash
mkdir $OWS_IMG_DIR/web_tmp
chmod 777 $OWS_IMG_DIR/web_tmp
```

### 2.2.6.3 Making the Graphics Builder web cartridge available to end users

Once you have created displays that you want to deploy on the Web, you must provide a method for users to access them. To do so:

- Place your organization's displays in `ORACLE_HOME\GRAPHICS25\WEB_OGD` (e.g., `C:\ORANT\GRAPHICS25\WEB_OGD`). This makes them available to users through the pull-down list on the Graphics Web toolbar.

**Note:** On UNIX, you must place your displays in the directory set by the `GRAPHICS_WEB_DIR` environment variable.

- Inform users of the URL that accesses the cartridge you created. The URL is in the format:

  ```text
  http://your_server/your_virtual_path
  ```

  The end point of the URL corresponds to the cartridge value you specified in the Virtual Path field (when you created the cartridge entry in the Web Request Broker). For example, if you specified “/ogweb” as the Virtual Path, the URL would be:

  ```text
  http://your_server/ogweb
  ```
2.3 Guidelines for Designing Web Applications

This section covers these topics:

- Section 2.3.1, “General guidelines for designing Web applications”
- Section 2.3.2, “Guidelines for designing Form Builder Web applications”
- Section 2.3.3, “Guidelines for designing Report Builder Web applications”
- Section 2.3.4, “Guidelines for designing Graphics Builder web applications”

2.3.1 General guidelines for designing Web applications

- Seriously consider network factors that affect the performance of your Web applications (e.g., interaction with security firewalls, heavy user loads, frequent network roundtrips to application and database servers, and other factors).

- Limit the number of image items and background images you include in your forms and reports. Each time an image is required, it must download from the application server. To display a company logo, for example, try including the image in the HTML page that downloads at application startup instead of retrieving it from the database or filesystem. For information customizing the startup HTML page, see Section 2.3.1.2, “Creating your own template HTML files” at the end of this chapter.

- Optimize your network connections where possible. Refer to your SQL*Net documentation for hints on enhancing SQL*Net performance.

- Design your queries to execute as efficiently as possible, and ensure PL/SQL program units are compiled. For more information, refer to the Oracle7 Server Concepts Manual and the PL/SQL User’s Guide and Reference.

2.3.1.1 Using RUN_PRODUCT to integrate Developer/2000 applications

To invoke Report Builder or Graphics Builder from a Web-enabled form, use the Form Builder built-in subprogram RUN_PRODUCT.
2.3.1.1.1 Running Report Builder Reports from a Web-enabled Form Builder application

To use RUN_PRODUCT to run a report from a form running on the Web, you must set three environment variables:

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FORMS50_OUTPUT</td>
<td>Physical directory on the application server in which to store generated Report Builder files. For example: <code>c:\orant\forms50\my_reps\</code></td>
</tr>
<tr>
<td>FORMS50_MAPPING</td>
<td>Virtual directory pointing to the physical directory defined by the FORMS50_OUTPUT variable. For example: <code>/web_reps/</code></td>
</tr>
<tr>
<td>FORMS50_REPFORMAT</td>
<td>Format in which to store generated Report Builder output. For example: PDF or HTML</td>
</tr>
</tbody>
</table>

**Table 2-2  Environment variables for using RUN_PRODUCT in web applications**

**Note:** On Windows NT, you define your environment variables in the Registry. On UNIX, you define your environment variables in the command shell. For more information on setting up environment variables, refer to the Developer/2000 installation instructions for your operating system.

After you set the environment variables above, the following sequence occurs automatically when a form running on the Web calls RUN_PRODUCT to invoke Report Builder:

If the output format of the report is FILE:

- the report does not display in the user’s browser
- the resulting file is stored in the physical directory specified by the FORMS50_OUTPUT environment variable
- the filename of the report file is the same name that is defined in the form definition

If the output format of the report is SCREEN or PREVIEW:

- the resulting output is stored (as a temporary file with an auto-generated filename) in the physical directory specified by the FORMS50_OUTPUT environment variable
- the Web server looks for the temporary filename (in the virtual directory defined by the FORMS50_MAPPING environment variable)
the Web server checks the desired display format specified by the FORMS50_REPFORMAT environment variable, and displays the report in that format in the end user’s browser.

Note: For detailed information on RUN_PRODUCT and its syntax, refer to the “RUN_PRODUCT” topic in the Form Builder online help.

2.3.1.1.3 Running Graphics Builder from a Web-enabled Form Builder application If your Web-enabled forms call RUN_PRODUCT to show a Graphics Builder display, you do not have to set any special environment variables.

2.3.1.2 Creating your own template HTML files
Consider creating your own static and cartridge HTML file templates (by modifying the templates provided by Oracle). By doing this, you can hardcode standard Forms Client applet parameters and parameter values into the template. Your template can include standard text, a browser window title, or images (such as a company logo) that would appear on the first Web page users see when they run Web-enabled forms. Adding standard parameters, values, and additional text or images reduces the amount of work required to customize the template for a specific application. To add text, images, or a window title, simply include the appropriate tags in the template HTML file.

For more information, see Section A.3, “Working with the Static (Non-Cartridge) HTML File Template” and Section A.2, “Working with the Cartridge HTML File Template”.

2.3.1.3 Creating an HTML application menu
As you deploy more and more Form Builder and Report Builder applications on the Web, try creating a single HTML page to serve as a centralized menu for your various Web-enabled applications. This approach eliminates the need to broadcast the URL of every application you deploy or remove. As you change your roster of available applications, simply modify the collection of links on the Web menu. End users then contact the menu URL and select from the list of available forms and reports.
2.3 Guidelines for Designing Web Applications

2.3.2 Guidelines for designing Form Builder Web applications

When designing Form Builder applications for the Web, consider the guidelines discussed in:

- Section 2.3.2.1, “Using Designer/2000 with the Developer/2000 Server”
- Section 2.3.2.2, “Reducing network traffic”
- Section 2.3.2.3, “Creating a generic GUI for Web-enabled Form Builder applications”
Section 2.3.2.4, “Selecting fonts”
Section 2.3.2.5, “Using the SHOW_DOCUMENT built-in”
Section 2.3.2.6, “Working with images”
Section 2.3.2.7, “Feature restrictions for forms on the Web”

2.3.2.1 Using Designer/2000 with the Developer/2000 Server

the Developer/2000 Server supports forms generated by Designer/2000 (32-bit, Release 1.3.2 or higher). If you use the standard Designer/2000 forms generator templates (ofg4pc1t.fmb and ofg4pc2t.fmb) to generate form and menu definitions, you can use the Developer/2000 Server to compile .FMX and .MMX files and immediately run the applications on the Web.

2.3.2.2 Reducing network traffic

To cut down on the number of network roundtrips required for users to operate your Form Builder applications on the Web, consider reducing or eliminating the following Form Builder features in your applications:

- **Mouse triggers.** Including When-Mouse-Click, When-Mouse-DoubleClick, When-Mouse-Down, and When-Mouse-Up triggers in your forms will impact speed and performance; the Forms Client must communicate with the Forms Server (necessitating a network roundtrip) each time one of these trigger fires. The When-Mouse-Move trigger is not supported due to the high number of network roundtrips required each time it fires.

- **Timers.** If your form includes a timer that fires every 1/100th of a second, end users face the performance ramifications of 60,000 network roundtrips every minute. Either reduce the number of timers in your forms, or change the timing interval on which your timers fire.

- **Image items and background images.** See Section 2.3.2, “Guidelines for designing Form Builder Web applications”.

2.3.2.3 Creating a generic GUI for Web-enabled Form Builder applications

Any end user with a Java-enabled Web browser can run your Web-enabled Form Builder applications. To make sure the application’s appearance does not vary from end user to end user—from Web browser to Web browser—you must create a GUI that is generic across platforms.
2.3.2.4 Selecting fonts

Most fonts are not supported across all platforms. For example, San Serif is a commonly-used font in Microsoft Windows applications. However, San Serif is not available in UNIX. When a font is not available on a platform, Form Builder attempts to use a similar font. As a result, when designing forms to deploy on the Web, be sure to follow the font guidelines below.

At runtime, the Forms Server maps a form’s fonts into their Java equivalents. Java then renders the font in a font pre-defined for the deployment platform. To convert your form’s fonts into Java equivalents, Java uses an alias list, located in the file called Registry.dat.

The following table lists the Java fonts, and their equivalents on the major deployment platforms:

<table>
<thead>
<tr>
<th>Java Font</th>
<th>Windows Font</th>
<th>X Windows Font</th>
<th>Macintosh Font</th>
</tr>
</thead>
<tbody>
<tr>
<td>Courier</td>
<td>Courier New</td>
<td>adobe-courier</td>
<td>Courier</td>
</tr>
<tr>
<td>Dialog</td>
<td>MS Sans Serif</td>
<td>b&amp;h-lucida</td>
<td>Geneva</td>
</tr>
<tr>
<td>DialogInput</td>
<td>MS Sans Serif</td>
<td>b&amp;h-lucidatypewriter</td>
<td>Geneva</td>
</tr>
<tr>
<td>Helvetica</td>
<td>Arial</td>
<td>adobe-helvetica</td>
<td>Helvetica</td>
</tr>
<tr>
<td>Symbol</td>
<td>WingDings</td>
<td>itc-zapfdingbats</td>
<td>Symbol</td>
</tr>
<tr>
<td>TimesRoman</td>
<td>Times New Roman</td>
<td>adobe-times</td>
<td>Times Roman</td>
</tr>
</tbody>
</table>

Note: If a font from your form does not map to a Java font (through the Form Builder font alias table), Java automatically assigns a Java font to the unmapped application font. To change the Java font alias scheme, simply edit Registry.dat.

2.3.2.5 Using the SHOW_DOCUMENT built-in

You can display a Web page from within a Web-deployed Form Builder application via the SHOW_DOCUMENT built-in. This built-in enables you to specify the URL of a Web page, and where you want to display the Web page.

For information on the SHOW_DOCUMENT built-in and its syntax, refer to the Form Builder online help.
2.3.2.6 Working with images

Wherever possible, reduce the number of image items and background images displayed in your applications. Each time an image is displayed to application users, the image must be downloaded from the application server to the end user’s Web browser.

To display a company logo with your Web application, include the image in the HTML file that downloads at application startup (instead of including it as a background image in the application, where it must be retrieved from the database or filesystem and downloaded repeatedly to users’ machines). For information on customizing the startup HTML page, see Section 2.3.1.2, “Creating your own template HTML files”.

2.3.2.7 Feature restrictions for forms on the Web

When designing forms for eventual deployment on the Web, keep in mind that certain Form Builder features behave differently—or not at all—when a form is deployed on the Web. The following table lists Form Builder features, whether the feature is supported on the Web, and any notes or guidelines about the feature:

<table>
<thead>
<tr>
<th>Feature</th>
<th>Supported?</th>
<th>Guidelines and Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActiveX, OCX, OLE, VBX</td>
<td>No</td>
<td>Third-party controls that display screen output on the application server are not supported because end users cannot view the output.</td>
</tr>
<tr>
<td>combo boxes</td>
<td>No</td>
<td>Available in Developer/2000 Release 2.1, pending Java implementation of combination box widget.</td>
</tr>
<tr>
<td>console</td>
<td>Yes</td>
<td>To display the console (includes the status and message lines) to end users, set the form-level property Console Window to the window in which you wish to display the console.</td>
</tr>
<tr>
<td>firewall</td>
<td>Yes</td>
<td>To deploy Form Builder applications on the Internet, you must run the Forms Server outside a security firewall (if any), and you must have a tunnel (through the firewall) that supports SQL*Net.</td>
</tr>
<tr>
<td>HOST_COMMAND, ORA_FFI, USER_EXIT</td>
<td>Yes</td>
<td>Calls to these functions often display visual output or GUI elements on end users’ machines in client/server mode. In a Web implementation, the same calls will display the output and GUI elements on the application server (where end users cannot see or interact with them).</td>
</tr>
</tbody>
</table>
2.3 Guidelines for Designing Web Applications

2.3.3 Guidelines for designing Report Builder Web applications

This section covers the following topics:

- Section 2.3.3.1, “About Web reports”
- Section 2.3.3.2, “Adding Web functionality”
- Section 2.3.3.3, “Selecting the output format”
- Section 2.3.3.4, “Helpful tips”

<table>
<thead>
<tr>
<th>Feature</th>
<th>Supported?</th>
<th>Guidelines and Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>iconic buttons</td>
<td>Yes</td>
<td>Icon image files must be in GIF format (and not in ICO format), and must reside in the HTML page or Web cartridge.</td>
</tr>
<tr>
<td>MDI (window and toolbars)</td>
<td>No</td>
<td>If the console or any toolbars are assigned to the Multiple Document Interface (MDI) window, you must re-assign them to another window in your Form Builder application.</td>
</tr>
<tr>
<td>NLS, BIDI</td>
<td>Yes</td>
<td>Supported for 8-bit languages only.</td>
</tr>
<tr>
<td>When-Mouse-Enter/Leave/Move triggers</td>
<td>No</td>
<td>Each execution of the trigger requires a network roundtrip, which would downgrade performance.</td>
</tr>
</tbody>
</table>

2.3.3.1 About Web reports

Report Builder enables you to generate files for display on the Web in the following formats:

- HTML can be displayed in an HTML 3.0-compliant Web browser, such as Netscape version 2.2 or higher.
- HTML with cascading style sheets can be displayed by an HTML 3.0-compliant Web browser that supports cascading style sheets (e.g., Microsoft Internet Explorer 3.01).
- PDF can be displayed in a standalone or plug-in PDF viewer, such as Adobe Acrobat Reader.

While it is possible to generate HTML or PDF output without changing your report definition, the report is often more useful on the Web if you add Web functionality (e.g., bookmarks and hypertext links) to it. The Web Wizard in Report Builder is the quickest way to prepare a report for the Web. It enables you to add basic Web
functionality and generate HTML or PDF output immediately. If you want to add further Web functionality or tweak what the Web Wizard has done, you can use the Report Editor and/or the Object Navigator.

Note: Web functionality in a report is ignored when you are not generating HTML or PDF. Adding Web functionality will not interfere with your ability to run the report to more traditional destination types (printers, ASCII files, etc.).

2.3.3.2 Adding Web functionality

To make your report more useful on the Web, Report Builder provides the capability to add Web functionality to it. You can add Web functionality to a report in any of the following ways:

- **Web Wizard**  In the Web Wizard, you can specify bookmarks and HTML files to insert at the beginning and end of the report.

- **Property Palette**  In the Property Palette for the report, you can enter Report Escapes that define HTML to be inserted at the beginning and end of the report, and at the beginning and end of pages. In the Property Palette for layout objects, you can specify Web Link properties that make the object a hyperlink destination or hyperlink, or execute a PDF application command line.

- **SRW Package**  The SRW package provides procedures that set Web properties programatically in triggers. For example, you could use a procedure from the SRW package in the format trigger of a layout object to make it a hyperlink.

2.3.3.2.1 Hyperlinks  Hyperlinks (or hypertext links) allow users to navigate to different sections of a report or other documents on the Web. To create a hyperlink, you first create a target (or hyperlink destination), which is a unique identifier for the current location. Next, you create a hyperlink elsewhere in the same or a different document that points to the target. When users click on the hyperlink, they will be taken to the target to which it points.

A hyperlink can take you to:

- another document on the same machine. For example:
  
  file:/private/mynewdoc.pdf or file:///C|/temp/mynewdoc.pdf

- another document on a different machine. For example:
  
  http://www.newmach.com/newdoc.pdf

- a destination within the current document. For example:
  
  #my_dest_name

- a destination within a local document. For example:
2.3 Guidelines for Designing Web Applications

- a destination within a remote document. For example:
  http://www.newmach.com/newdoc.pdf#some_dest_name
- any URL. For example:
  http://www.newmach.com/newdoc.html
  ftp://www.reposit.com/filetoget.example
  http://www.someach.com/cgi-bin/webmenu?choice1

In the example below, a top-level report contains hypertext links that display more detailed reports for each region. A hypertext contact address displays an e-mail template for sending comments to the Webmaster.

![Example of a hypertext report](image-url)

*Figure 2–2 Example of a hypertext report*
Example 2–1  Hyperlink PL/SQL example

The format trigger below defines a hypertext link to a destination (mytarget) in another document (newdoc.pdf) for the boilerplate object B_2.

Note: If the target were in the same document, you would omit http://www.newmach.com/newdoc.pdf.

function B_2FormatTrigger return boolean is
begin
    srw.set_hyperlink('http://www.newmach.com/newdoc.pdf' ||
                     '#mytarget');
    return (TRUE);
end;

Example 2–2  Dynamic hyperlink example

The format trigger below defines a hypertext link for the boilerplate object B_3. Notice how the destination of the link is determined dynamically based upon the values of SRCDIR and CF_PICKVIDEO. For example, if the value of SRCDIR is http://www.newmach.com, and the value of CF_PICKVIDEO is good.avi, this function would assign the following hypertext link to the object:

http://www.newmach.com/webdemo/src/good.avi

Format trigger:

function B_3FormatTrigger return boolean is
begin
    srw.set_hyperlink(:srcdir||:cf_pickvideo);
    if ( upper(:cf_pickvideo) like '%GOOD%' ) then
        srw.set_foreground_bordercolor('green');
        srw.set_background_border_color('green')
    end if;
    return (TRUE);
end;

/* SRCDIR is a parameter whose value is determined at ** runtime by the following After Parameter Form Trigger */

function AfterPForm return boolean is
begin
    :srcdir := :web_server || '/webdemo/src/';
2.3 Guidelines for Designing Web Applications

```
:docsdir := :web_server || '/webdemo/docs/';
return (TRUE);
end;
/* CF_PICKVIDEO is a formula column whose value is
** determined by the following function
*/
function CF_pickvideoFormula return Char is
begin
  if (:avg_h_div < .80 )
    then return ('bad.avi');
    else return ('good.avi');
  end if;
end;
end;

2.3.3.2.2 Bookmark

A bookmark is a string that will appear in a frame of the master HTML document or in the PDF viewer if you open the bookmark area. Clicking on the bookmark displays the associated object at the top of the window. A list of bookmarks can serve as an outline or Table of Contents for your report.

For reports with master-detail relationships, the Web Wizard can create bookmarks for you automatically. You can also create bookmarks by specifying the Bookmark property via the Property Palette or `SRW.SET_ATTR` in a format trigger. In the example below, a PDF report contains bookmarks that correspond to different sections of the report.

Figure 2–3 Example of a bookmark report

By default, the order in which bookmarks appear in the list is determined by when Report Builder renders the objects when you generate the report. You can hierarchically order bookmarks by embedding ordering information in the bookmark definition string. A string with explicit ordering information takes the form:

```
x#book_mark_name
```

where x is an outline number. The pound sign (#) and outline number do not appear in the bookmark area but are used to determine order and indentation. For example:

```
1#Expense Summary Section
2#Expense Detail Section
2.1#Expenses for the Administration Department
2.2#Expenses for the Engineering Department
```
2.3 Expenses for the Sales Department
  2.3.1 Expenses for the Eastern Sales Region
  2.3.2 Expenses for the Northern Sales Region
  2.3.3 Expenses for the Southern Sales Region
  2.3.4 Expenses for the Western Sales Region

**Example 2–3  Example of bookmark with ordering information**

The format trigger below defines a bookmark for the boilerplate object B_2. Note the use of explicit ordering information (1#) in this example. If you did not want the ordering information, you could omit 1#.

```plaintext
function B_2FormatTrigger return boolean is
begin
  srw.set_bookmark('1#Expense Summary Section');
  return (TRUE);
end;
```

**Example 2–4  Example of dynamic bookmark**

The format trigger below defines a bookmark for the boilerplate object B_2. Notice that the name of the bookmark is dynamic. CATEGORY is a column value that is concatenated with the string Expense Summary Section for each execution of the Format Trigger. In this case, CATEGORY could contain ordering information (for example, 1#) or perhaps a string that makes the bookmark unique within the report.

```plaintext
function B_2FormatTrigger return boolean is
begin
  srw.set_bookmark(:category || '
    ' Expense Summary Section');
  return (TRUE);
end;
```

**2.3.3.2.3 Action  (PDF only)** An action is a command line (for example, `c:\orawin\bin\r30run userid=scott/tiger report=example.rdf` or `/usr/local/bin/phone smith`) that will be executed on the local machine when the object is clicked in the PDF viewer.

**Note:** An object that is associated with an action cannot also be the source of a hypertext link.

**2.3.3.2.4 Report Escapes  (HTML Only)** Report Escapes enable you to insert HTML code at certain points in your report output. This feature is useful for such things
as adding consistent headers/footers to each page or giving all of you HTML reports a consistent look and feel. You can enter the HTML code you want to insert directly or as a file.

Report Escapes can be entered in the Property Palette for the report or using the SRW packaged procedures. The Report Escapes are listed below with their SRW packaged counterparts in parentheses:

- **Before Report** *(SRW.SET_BEFORE_REPORT_HTML)* The Before Report escape inserts the specified text, graphics, or HTML commands at the beginning of your document. This attribute is useful for placing a logo or some standard links at the beginning of an HTML document.

- **After Report** *(SRW.SET_AFTER_REPORT_HTML)* The After Report escape inserts the specified text, graphics, or HTML commands at the end of your document. This attribute is useful for placing a logo or some standard links at the end of an HTML document.

- **Before Page** *(SRW.SET_BEFORE_PAGE_HTML)* The Before Page escape inserts the specified text, graphics, or HTML commands at the beginning of pages of your document. This attribute is useful for placing a logo or some standard links at the beginning of each page in an HTML document.

- **After Page** *(SRW.SET_AFTER_PAGE_HTML)* The After Page escape inserts the specified text, graphics, or HTML commands at the end of pages of your document. This attribute is useful for placing a logo or some standard links at the end of each page in an HTML document.

- **Before Form** *(SRW.SET_BEFORE_FORM_HTML)* The Before Form escape inserts the specified text, graphics, or HTML commands at the beginning of your runtime parameter form.

- **After Form** *(SRW.SET_AFTER_FORM_HTML)* The After Form escape inserts the specified text, graphics, or HTML commands at the end of your runtime parameter form.

For more information on how to add Web attributes to reports, refer to the Report Builder online help system.

**2.3.3.3 Selecting the output format**

There are several differences between HTML and PDF output. Understanding the differences will help you to choose the best format for your Web reports.
2.3.3.3.1 About HTML output  In general, HTML reports display faster because they are in ASCII format. Here’s an example of an HTML report:

![HTML report example](image)

*Figure 2–4  Example of an HTML report*
2.3 Guidelines for Designing Web Applications

The fidelity of HTML output to your printed output is not as high as with PDF format. Keep in mind the following:

**Backgrounds**

Background (fill) and border (line) colors/patterns for text are not available in HTML. Bold, italic, underline, and foreground (text) color are supported if the browser supports them.

**Bookmarks**

Bookmarks cause multiple HTML files to be created. One master file is created with two frames, one for bookmarks and one for the report output. One HTML file is created for each of these frames. The master document filename is the name specified in the DESNAME parameter. The bookmark filename is `<desname>b.htm`. The report output filename is `<desname>d.htm`.

**Buttons**

Buttons with text labels are converted to text. Buttons with icons are converted to GIF files. In either case, the format trigger for the button is executed (for example, you could create a hypertext link for the button in the format trigger using `SRW.SET_HYPERLINK`). The action associated with the button (such as play multimedia or execute PL/SQL) is not preserved. For example, if a button is supposed to play a multimedia file, the video is not played when you click the button in the HTML output. The only way to have the button be “active” in HTML output is to use SRW packaged procedures in its format trigger.

**Drawings**

The only drawn object supported in HTML is a solid, black, horizontal line. The line width specified in the report may be honored depending upon the browser. All other drawn objects (for example, rectangles or circles) in the report layout will not show up in the HTML output. Space for these drawn objects is reserved, but there is no visible representation in the HTML output.

**Fonts**

HTML has seven sizes for text. However, the user can override the size specified in the HTML file from their browser. If the font used in the report is non-proportional, TeleType mode is turned on for the generated HTML file.

**Frames**

Report frames are not visually represented in the HTML output. Any fill or border attributes of frames do not appear in HTML output.
2.3 Guidelines for Designing Web Applications

Images

Linked images, image fields, and charts in a report cause GIF files to be created and referenced from the HTML document. Note that even if the linked boilerplate or image field refers to an external GIF file, a new GIF file is generated. If an image is stored in the database, one GIF file may be generated for each occurrence of the image in the report. If an image is stored in a file (for example, imported images, linked boilerplate, or image fields that reference files), only one GIF file will be generated per image regardless of how many times it is repeated in the report. Any GIF image files generated for HTML output have a number sequence (for example, desname>0.gif ... desname>17.gif).

OLE objects

When generating HTML output, OLE objects are ignored, but space is reserved in the output for the objects.

Overlapping objects

Objects cannot overlap one another. For example, you could not have text on top of an image. If objects overlap slightly (two characters or less), then the object underneath is truncated to prevent overlap. If objects overlap significantly (or one is completely on top of the other), then the underneath object is removed altogether. In this case, any linking information of the removed object is transferred using the same rules as if it were a frame. Text always takes precedence over horizontal lines, regardless of which is on top. This prevents the line underneath a column label from eliminating the label text.

Pagination

HTML does not have the concept of a page. A separator line is placed between each page of the report. Depending upon the browser, you may or may not need to scroll to see the entire report page. Furthermore, if you print the HTML document from your browser, the printer will not necessarily print the separator lines at the bottom of each page. If you do not want the separator line or you want to use a different separator line, you can use the After Page escape to change it.
2.3.3.2 About PDF format In general, PDF output results in high-fidelity reports and the convenience of a single output file. Here’s an example of a PDF report:

![Example of a PDF report](http://webdemo/out/trades.pdf)

**Figure 2–5 Example of a PDF report**

PDF reports can take longer to display and are less portable, requiring a PDF viewer or plug-in to view. Here are some additional differences:
Buttons

Buttons with text labels are converted to text. Buttons with icons are converted to GIF files. In either case, the format trigger for the button is executed (for example, you could create a hypertext link for the button in the format trigger using \texttt{SRW\_SET\_HYPERLINK}). The action associated with the button (such as play multimedia or execute PL/SQL) is not preserved in the PDF output. For example, if a button is supposed to play a video, the video will not be played when you click the button in PDF output. The only way to have the button be “active” in PDF output is to use SRW packaged procedures in its format trigger.

OLE objects

When generating PDF output, OLE objects are ignored, but space is reserved in the output for the objects.

2.3.3.4 Helpful tips

- The Reports Web Cartridge or CGI may create temporary files for report output that you may need to delete periodically. They will be located in the path specified by \texttt{REPORT30\_WEBLOC\_TRANSLATED}.
- For more information, see the demo Web reports, or refer to the Report Builder online help system.

2.3.4 Guidelines for designing Graphics Builder web applications

The following are tips for the efficient deployment of Graphics Builder displays on the Web:

- Keep display file sizes small:
  - Limit the number of layers in a display.
  - Create objects programmatically.
  - Take advantage of stored procedures for data-intensive displays.
- Put as much code as possible into libraries to maximize code sharing between objects and applications and minimize file size during loading.
- Limit the use of multimedia within an application. Where you do use multimedia, redefine button triggers to make a call to a URL that contains media information.
- Limit activities that involve screen refreshing. For example, avoid the use of user interface-based timers. Design user interfaces so that events are initiated
by user interaction rather than by timers.

- Take advantage of custom hyperlinks to create hyperlink drill-downs. For more information about custom hyperlinks, see Section 2.3.4.1, “Using custom hyperlinks in Graphics Builder displays”.

- Avoid the use of the following unsupported features when designing Graphics Builder displays for the Web:
  - Drag-and-drop.
  - Mouse_Up trigger events. Only Mouse_Down is supported for this release.
  - Multi-window applications. Instead, use layers rather than multiple windows, or use custom hyperlinks to link multiple displays in separate windows. For more information about using custom hyperlinks in Graphics Builder displays, see Section 2.3.4.1, “Using custom hyperlinks in Graphics Builder displays”.

2.3.4.1 Using custom hyperlinks in Graphics Builder displays

the Developer/2000 Server provides the capability to enhance your Graphics Builder displays through the use of custom hyperlinks. Custom hyperlinks might be used, for example, to design clickable links to mail messages. Or they might be used to design multi-frame HTML pages with clickable Tables of Contents that update the contents of adjoining frames. See the sample files displays provided with the Graphics Server for examples of how you can use custom hyperlinks in Graphics Builder displays.

To include custom hyperlinks in a Graphics Builder display, first create the object that will function as the hyperlink (for example, an underlined text item or a rectangle). In the object’s button procedure code, set the following parameters:

```latex
og_set_param ('OG_URL', 'http://my_url.com');
```

and, optionally:

```latex
og_set_param ('OG_URL_TARGET', my_target);
```

where `my_target` is one of the following text arguments:

- `(empty string)` The URL specified replaces the contents of the current browser window (default behavior).
- `frames [n]` The URL specified displays within the nth frame in the current page.
The URL specified displays within a new browser window.

**Note:** The `OG_URL_TARGET` parameter is optional. If it is omitted, the URL specified in `OG_URL` is displayed within the current browser window.
2.3 Guidelines for Designing Web Applications
This chapter offers guidelines to help you develop a graphical user interface (GUI) with Developer/2000:

<table>
<thead>
<tr>
<th>Section Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Section 3.1, “Understanding the Process”</td>
</tr>
<tr>
<td>Section 3.2, “Creating an Effective Form”</td>
</tr>
<tr>
<td>Section 3.3, “Creating an Effective Report”</td>
</tr>
<tr>
<td>Section 3.4, “Creating an Effective Display”</td>
</tr>
</tbody>
</table>

### 3.1 Understanding the Process

Even more important than understanding the process for developing an effective GUI is understanding the people who will use it. In fact, your success is directly related to how well you understand your users—the tasks they perform, the order in which they perform them, their surroundings, and their expectations.
If you’re like many application developers, this idea may require a profound shift of focus. Applications typically evolve from the inside out: from the datasource itself, to the code, and finally to the GUI. If you are committed to developing an effective GUI, you must reverse this process: first, interview your users; next, design a user interface that supports their specific tasks; and finally, create the underlying code base that makes it all work.

**Figure 3–1  Thinking about the user first**

No set of prepackaged standards or guidelines can serve as a substitute for developing an accurate understanding of your users’ needs. This chapter can help you develop that understanding, as well as assist you in creating an interface uniquely tailored to your particular group of users.
3.1.1 What are the stages?

As shown in the Figure 3–2, the process for developing a GUI consists of four major stages:

- **Define User Requirements**
  - Gather documentation
  - Interview users
  - Observe users
- **Plan the User Interface**
  - Produce process plans
  - Plan UI elements
  - Plan/implement standards mechanisms
- **Build the User Interface Elements**
  - Create/modify UI elements
  - Create/modify UI functionality
- **User Feedback and Testing**
  - Prototype testing and monitoring
  - Apply user feedback

**Figure 3–2 Stages in developing a user interface**

The rest of this section offers guidelines for completing each of these stages:

- Section 3.1.2, “Defining user requirements”
- Section 3.1.3, “Planning the user interface”
- Section 3.1.4, “Building the user interface elements”
- Section 3.1.5, “Gathering user feedback”

**Note:** This chapter is not intended to treat the subject of GUI development exhaustively. If you require more detail on how to proceed in a given stage, you may want to visit your local library or computer bookstore. In particular, Jeffrey Rubin’s “Handbook of Usability Testing” is an excellent source of information on defining user requirements and gathering user feedback.

3.1.2 Defining user requirements

In the first stage of GUI development, you determine what the user needs and expects from your application. While it may be tempting to skip this stage and move right to the design phase, it's risky to do so. Without a clear understanding of the users themselves and the tasks they must perform, it is virtually impossible to create an effective GUI.
To define user requirements:

- **Gather documentation.** Relevant policies and procedures manuals and existing documentation about the system (whether previously computerized or not) will help you formulate the necessary background for conducting user interviews.

- **Observe users doing their jobs.** Make a list of the tasks users perform and the order in which they perform them.

- **Interview users.** Find out what people want from a GUI-based system. When conducting your interviews:
  - Ask not only what users do, but how they work. For example, does a clerk need to be able to work on several orders at the same time, or just on one?
  - Find out what users like and don’t like about the current system (even if it’s not computerized).
  - Ask users how they envision the GUI. Encourage them to provide as much detail as possible.
  - Get to know the users. Do users typically stay on the job for a long time or is there high turnover? Will they use the application constantly or only occasionally? For infrequently used applications, you’ll want to provide a lot of buttons, text, and guidelines to help reduce the amount of familiarization time. For applications that are used daily, try to provide a lot of shortcuts and accelerator keys to help experienced users complete their tasks quickly.
  - Find out if users have any disabilities or special circumstances you should consider. For example, are users typically standing when they use the application? If so, they won’t have the time or patience for excessive navigation.

- **Sample a wide variety of users.** Feedback from users at a single customer site are biased toward their specific experiences.

### 3.1.3 Planning the user interface

In the second stage, you plan and document how you will implement a user interface that meets the users’ needs. This involves:

- Developing a set of standards that you will adhere to and, if necessary, obtaining buy-in from your team. Refer to Section 3.1.3.1, “Creating your standards”.
• Considering platform-specific requirements and other restrictions in the deployment environment. Refer to Section 3.1.3.2, “Considering portability”.

• Mapping out each screen and deciding which types of interface elements to use in order to meet user needs effectively. Refer to Section 3.1.3.3, “Creating a prototype”.

3.1.3.1 Creating your standards

A set of consistent development standards is crucial to the success of any development effort. By developing and enforcing standards pertaining to layout, use, and behavior of various GUI elements, you can ensure that even disparate parts of the application have a common look and feel. Developer/2000 offers several mechanisms to assist you in developing a consistent set of standards.

<table>
<thead>
<tr>
<th>Mechanism</th>
<th>Description</th>
</tr>
</thead>
</table>
| Object Library (Form Builder)    | An object library is a set of objects and standards that you create and make available to your entire development team. Through the use of subclassing, each developer can ensure that changes made to the objects in the object library are propagated throughout all applications that use them. **Object libraries are the preferred mechanism for standardizing your Form Builder applications.** Form Builder provides two object libraries which you can customize to meet your own site requirements:  
  ■ Standard Object Library, which contains suggested standards optimized for the Windows 95 environment.  
  ■ Oracle Applications Object Library, which contains standards for cross-platform applications: Windows 95, Solaris, Macintosh, and character mode.  

For more information, see the Form Builder online help topics “About object libraries” and “About subclassing”.

*Table 3–1 Developer/2000 Standards Mechanisms*
### Object Group (Form Builder)

An object group is a container for a group of objects. You define an object group when you want to package related objects so you can copy or subclass them in another module.

For example, suppose you build an appointment scheduler using several types of objects, including a window and canvas, blocks, items that display dates and appointments, and triggers that contain the logic for scheduling and other functionality. By packaging these objects into an object group, you can copy all of them to other forms in one simple operation.

For more information, see the Form Builder online help topic “Guidelines for using object groups”.

### Visual attributes (Form Builder)

Visual attributes are the font, color, and pattern properties you set for form and menu objects that appear in your application’s GUI. Visual attributes can include the following properties:

- Font properties: Font Name, Font Size, Font Style, Font Width, Font Weight
- Color and pattern properties: Foreground Color, Background Color, Fill Pattern, Charmode Logical Attribute, White on Black

For more information, see the Form Builder online help topic “Guidelines for using visual attributes”.

### Template (Form Builder, Report Builder)

In Form Builder, you can create templates to provide other team members with a default starting point for new forms. Templates typically include generic objects, such as graphics (like corporate logos), toolbars, program units, standard window layouts, toolbars, and menus, and other common objects.

Report Builder not only allows you to create your own templates to help control the appearance of your reports, but provides a wide variety of pre-defined templates as well. Using the Report Wizard, you select the objects you want to include in your report, then select a template to arrange those objects and apply standard formatting attributes.

For more information, search the Form Builder and Report Builder online help index for “templates”.

<table>
<thead>
<tr>
<th>Mechanism</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object Group (Form Builder)</td>
<td>An object group is a container for a group of objects. You define an object group when you want to package related objects so you can copy or subclass them in another module. For example, suppose you build an appointment scheduler using several types of objects, including a window and canvas, blocks, items that display dates and appointments, and triggers that contain the logic for scheduling and other functionality. By packaging these objects into an object group, you can copy all of them to other forms in one simple operation. For more information, see the Form Builder online help topic “Guidelines for using object groups”.</td>
</tr>
</tbody>
</table>
| Visual attributes (Form Builder) | Visual attributes are the font, color, and pattern properties you set for form and menu objects that appear in your application’s GUI. Visual attributes can include the following properties:  
  - Font properties: Font Name, Font Size, Font Style, Font Width, Font Weight  
  - Color and pattern properties: Foreground Color, Background Color, Fill Pattern, Charmode Logical Attribute, White on Black  
  For more information, see the Form Builder online help topic “Guidelines for using visual attributes”. |
| Template (Form Builder, Report Builder) | In Form Builder, you can create templates to provide other team members with a default starting point for new forms. Templates typically include generic objects, such as graphics (like corporate logos), toolbars, program units, standard window layouts, toolbars, and menus, and other common objects. Report Builder not only allows you to create your own templates to help control the appearance of your reports, but provides a wide variety of pre-defined templates as well. Using the Report Wizard, you select the objects you want to include in your report, then select a template to arrange those objects and apply standard formatting attributes. For more information, search the Form Builder and Report Builder online help index for “templates”.

---

Table 3-1  **Developer/2000 Standards Mechanisms**
3.1.3.2 Considering portability

If you intend to deploy your application in more than one environment, it’s important to understand how various GUI elements are rendered on each platform and which elements are restricted altogether. For example, due to formatting constraints between platforms, interactive buttons that you create for Windows may shrink and become less readable when displayed on Solaris. Chapter 5, “Designing Portable Applications”, helps you understand platform-specific constraints and provides tips and guidelines for working around them. It also provides considerations for character mode, which restricts the UI in numerous ways.

3.1.3.3 Creating a prototype

Prototypes are an extremely effective means for ensuring usability in your application. The most effective prototypes follow an iterative development model, beginning with a storyboard and ending with a fully functional application. The process breaks down as follows:

1. Draft a storyboard to give you a clear picture of how the application will actually look and behave. A storyboard is a frame-by-frame drawing of screens showing transition and appearance. Include a narrative to describe how the screens relate to the tasks you identified when you defined the users’ requirements.
Here is an example of three panels from a storyboard for an ordering application:

![Storyboard Example](image)

**Figure 3–3 Example of a storyboard**

2. Show the storyboard to users. Verify that your planned application addresses their needs and supports their tasks *the way they perform them*.

3. Expand the storyboard into a *paper prototype*. Whereas a storyboard sketches task and window flow at a high level, a paper prototype is a fairly detailed illustration of the entire application. A paper prototype typically contains one piece of paper for each window you’ve planned, complete with widgets, arrows to represent task flow and navigation, and so on.

4. Show the paper prototype to users. Most of the organizational issues should have been identified during the storyboard phase, so you can now focus on details: the placement of buttons, the layout of a supporting dialog, and so on. Section 3.1.5, “Gathering user feedback” offers some tips for conducting the session with users.
5. Based on user feedback, create a functional prototype using Developer/2000. The following sections can help you select the appropriate objects for your prototype:

- Section 3.2, “Creating an Effective Form”
- Section 3.3, “Creating an Effective Report”
- Section 3.4, “Creating an Effective Display”

6. Let users experiment with the functional prototype. Be sure to include users who were not involved in the earlier sessions so you can determine whether the application is easily grasped by new users.

7. Repeat steps 5 and 6 until you are satisfied that you have met all the objectives stated in your user requirements.

### 3.1.4 Building the user interface elements

Only when you have devoted sufficient time to developing your conceptual model—that is, when you fully understand your users and the tasks they perform and have designed smoothly flowing dialogs in support of those tasks—only then are you ready to begin building your user interface. This chapter contains three sections to help you choose your user interface elements carefully:

- Section 3.2, “Creating an Effective Form”
- Section 3.3, “Creating an Effective Report”
- Section 3.4, “Creating an Effective Display”

### 3.1.5 Gathering user feedback

When you have developed a working prototype, either on paper or with Developer/2000, return to the users you interviewed in the first phase and let them experiment with it. To gather user feedback effectively:

- Produce instructions for user tests using a task-based approach.
- To ensure a broad perspective, use at least six typical users.
- Record user activity through notes, sound, and video monitoring.
- Question users about the prototype’s performance.
- Get more than one designer to interpret the results.

Remember: only the actual user of your application is qualified to comment if the UI is appropriate.
After testing the prototype on users and gathering their feedback, return to the build stage, modify the user interface accordingly, then test your changes again. Continue this cycle until the interface meets the objectives you outlined in the requirement definition phase.

### 3.2 Creating an Effective Form

This section explains how to build an effective GUI using Form Builder.

**Note:** The information in this section assumes a Eurocentric viewpoint. (If you are developing for a non-Western audience, be sensitive to the cultural background of the users. If practical, have your design reviewed by several members of your target audience.)

### 3.2.1 Understanding forms

Before addressing specific considerations for forms, it may be helpful to briefly introduce some basic forms concepts. (Experienced Form Builder users should go to Section 3.2.2, “Guidelines for building forms”.) For more details on these and other related forms topics, see the Form Builder online help and/or the Form Builder Developer/2000 Quick Tour.

#### 3.2.1.1 What is a module?

When you build an application with Form Builder, you work with individual application components called *modules*. There are four types of modules in Form Builder:

<table>
<thead>
<tr>
<th>Module Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Form module</td>
<td>A collection of objects and code routines. Some of the objects you can define in a form module include windows, text items (fields), check boxes, buttons, alerts, lists of values, and blocks of PL/SQL code called triggers.</td>
</tr>
<tr>
<td>Menu module</td>
<td>A collection of menus (a main menu object and any number of submenu objects) and menu item commands.</td>
</tr>
<tr>
<td>PL/SQL Library module</td>
<td>A collection of user-named procedures, functions, and packages that can be called from other modules in the application.</td>
</tr>
</tbody>
</table>

*Table 3–2  Form Builder Modules*
3.2 Creating an Effective Form

This chapter does not address the use of PL/SQL library modules. For information on this topic, refer to the Form Builder online help.

3.2.1.2 What are forms, blocks, items, regions, and frames?

Simply put, a form (or form module) is an application that provides access to information stored in a datasource. When you look at a form, you see interface items such as check boxes, radio groups, and so on, which enables the user to interact with the datasource. These interface items belong to a container called a block. In Figure 3–4, the fields Customer ID, First name, Title, and so on all belong to the same block.

<table>
<thead>
<tr>
<th>Module Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object Library module</td>
<td>A collection of objects that can be used to develop applications. See Table 3–1, “Developer/2000 Standards Mechanisms” for more information.</td>
</tr>
</tbody>
</table>

**Table 3–2  Form Builder Modules**

This chapter does not address the use of PL/SQL library modules. For information on this topic, refer to the Form Builder online help.

3.2.1.2 What are forms, blocks, items, regions, and frames?

Simply put, a form (or form module) is an application that provides access to information stored in a datasource. When you look at a form, you see interface items such as check boxes, radio groups, and so on, which enables the user to interact with the datasource. These interface items belong to a container called a block. In Figure 3–4, the fields Customer ID, First name, Title, and so on all belong to the same block.

![Sample form](image)

**Figure 3–4  Sample form**

There are two types of blocks: a data block, which serves as a link between the datasource and the user, and a control block, which is not associated with a datasource.
Each data block can enable the user to view and access data from one table in the datasource. Blocks can be single-record blocks, which means that they show one row of data at a time, or multi-record blocks, which enable users to see many rows of data at once. All of the fields in Figure 3–4 are in single-record blocks.

A region is a rectangle or line that separates a logical grouping of fields from others in the block. In Figure 3–4, the rectangle that separates the Customer Information fields from the Accounts icons is a region.

A frame is a pre-defined way of arranging certain items in a block. For example, the block shown in Figure 3–4 was arranged by a frame that established its margins and offsets, the distance between the items and prompts, and so on.

### 3.2.1.3 What are windows and canvases?

A window is the container for all visual objects that make up a Form Builder application. A single form can include any number of windows; all but the simplest of forms have several windows associated with them. Several types of windows are available:

<table>
<thead>
<tr>
<th>Window type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Container (MDI)</td>
<td>Holds all other windows. It usually, but not always, contains the toolbar and main menu. (Windows only)</td>
</tr>
<tr>
<td>Modeless</td>
<td>Enables the user to interact with any other window, as well as the toolbar and the menu. Modeless windows are used most often in GUIs when the user is free to choose among many tasks.</td>
</tr>
<tr>
<td>Modal</td>
<td>Forces the user to work within a single window, then either accept or cancel the changes they have made. The toolbar and menu are not accessible. Use a modal window when the user must complete a particular task before continuing.</td>
</tr>
</tbody>
</table>

| Table 3–3 | Types of windows |
Here is an example of a typical Form Builder window:

**Figure 3–5  Typical Form Builder window**

Like most Windows 95 Form Builder windows, this one contains:

- Window title
- Menu bar and pull-down menus
- Buttons and other control items that do not correspond to the data
3.2 Creating an Effective Form

- Data items in the blocks
- Console, which includes the message line and status line

A canvas is the background object upon which interface items appear. There are four types of canvases:

<table>
<thead>
<tr>
<th>Canvas Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Content canvas</td>
<td>Occupies the entire pane of the window in which it is displayed (and possibly more, if the window enables scrolling). Every window has at least one content canvas.</td>
</tr>
<tr>
<td>Stacked canvas</td>
<td>Displayed atop—or stacked on—the content canvas assigned to the current window. Stacked canvases are useful for conditionally obscuring areas of the content canvas—unpopulated fields, for example. Through the use of viewports you can control how much of a stacked canvas is visible.</td>
</tr>
<tr>
<td>Tab canvas</td>
<td>A set of tabs that enable you to group and display a large amount of related information on a single dynamic canvas.</td>
</tr>
<tr>
<td>Toolbar canvas</td>
<td>Used to create toolbars for individual windows.</td>
</tr>
</tbody>
</table>

Each window may display one or more canvases. You can also conditionally display a canvas in a window, depending on whether certain conditions are met.

3.2.2 Guidelines for building forms

The following sections offer specific recommendations for building an effective GUI with Form Builder:

- Section 3.2.2.1, “Using object libraries”
- Section 3.2.2.2, “Understanding basic design principles”
- Section 3.2.2.3, “Adding color”
- Section 3.2.2.4, “Creating canvases”
- Section 3.2.2.5, “Creating windows”
- Section 3.2.2.6, “Creating regions”
- Section 3.2.2.7, “Adding items to blocks”
Section 3.2.2.8, “Designing messages”
Section 3.2.2.9, “Implementing online help”
Section 3.2.2.10, “Building effective menus”

3.2.2.1 Using object libraries

Perhaps the most important means of standardization available to you as a form developer is the object library. An object library is a set of objects and standards that you create; each object or standard can determine the appearance and layout of an entire frame, window, or region. When housed in an object library, these objects become available to all the developers on your project or site, thus ensuring that even developers working at different locations can produce an application—or different modules within the same application—with a common look and feel. Through the use of subclassing, each developer can ensure that changes made to the objects in the object library are propagated throughout all applications that use them.

A good strategy for using object libraries is to create a separate one for each logical grouping of standards. For example, you may want to have one object library for corporate standards that you make available company-wide, and another tailored for the specific needs of your project.

To help you get started building your own object libraries, Form Builder provides two samples:

- Standard Object Library, which contains objects for Windows 95-only deployments where multi-language support is not a requirement
- Oracle Applications Object Library, recommended for multi-platform deployments.

Before you create your own object library, it’s a good idea to test the contents of the Standard or Oracle Application Object Libraries to see what works well and what you need to modify.

<table>
<thead>
<tr>
<th>To test:</th>
<th>Do this:</th>
</tr>
</thead>
</table>
| The items in the Standard Object Library in a **data block** | 4. Use the Data Block Wizard to create a data block.  
5. Click an item in your control or data block, then click the right mouse button. A list of Smart Classes applicable to that item is displayed; click the SmartClass you want. |
3.2 Creating an Effective Form

If you use the Standard Object Library, be sure to subclass all the attributes under the VAGs tab to the Visual Attributes node in your form. Many of the standards are based upon these visual attributes and will not display correctly if the visual attributes are not applied. By subclassing (rather than copying) the visual attributes, you ensure that you always have access to the latest definitions.

If you know that you will be using a particular set of visual attribute groups in all or most of your forms, create a template form that already contains the visual attribute groups subclassed from the standard object library. Then you can name this template when prompted by the Layout Wizard.

For more information on the object libraries, see the help topics “Standard Object Library” and “Oracle Application Object Library” under “Reusable Components” in the Developer/2000 Demos online help.

### 3.2.2.2 Understanding basic design principles

Here are some general guidelines for building forms:

- Use a Real Coordinate system with a measurement unit of inches, centimeters, or points. Choose a single unit and use it across all modules.

<table>
<thead>
<tr>
<th>To test:</th>
<th>Do this:</th>
</tr>
</thead>
</table>
| The items in the Standard Object Library in a control block | 1. Open STNDRD20.OLB.  
2. Drag and drop the item(s) into the block. |
| Only the visual attributes in the Standard Object Library | Open the STNDRD20.OLB template form. |
| The objects in the Oracle Application Object Library | Open the APPSTDS.OLB template form. |

If you use the Standard Object Library, be sure to subclass all the attributes under the VAGs tab to the Visual Attributes node in your form. Many of the standards are based upon these visual attributes and will not display correctly if the visual attributes are not applied. By subclassing (rather than copying) the visual attributes, you ensure that you always have access to the latest definitions.

Points

Often the easiest to use, since you can designate sizes in whole numbers. Since text is always specified in points, it’s easier to size objects relative to text if the objects are in points as well.

Inches and centimeters

Enables you to specify a higher precision than points (but it’s not as easy to compare the size of objects to text). Useful if your target environment is exclusively SVGA or better.
### 3.2 Creating an Effective Form

- Place users in control by enabling them to enable or disable dialogs wherever it makes sense to do so. Making this determination requires you to carefully balance your knowledge of the users with the freedoms or restraints imposed by their working conditions.

  **Example:** All users should be enabled simple freedoms, like the ability to interrupt an application and resume it later on. But enabling users the ability to completely rearrange a company-issued invoice may not be wise, since it affords the user power to disregard company standards.

  The extent to which users should have control is also determined by the user’s experience level. If you are developing an application for both experienced and inexperienced users, consider providing a wizard to provide step-by-step assistance for those who want it, along with manual alternatives.

- Make it obvious to users when a task is finished, either by closing a window, opening another window, or displaying an informational message.

- Make windows only as large as necessary.

- Use blank space as a way to group information.

- Use the frame objects in the Standard Object Library to help you obtain a consistent layout style across form modules.

- Orient screen layouts based on a top-to-bottom task sequence. Arrange blocks, regions, and items in the order they will be used, from left-to-right, then top-to-bottom.

- In single-record blocks, left-align items where possible. (Right-align fields containing currency and numbers.) In multi-record blocks, stack items horizontally and align them along the top.

---

<table>
<thead>
<tr>
<th>Pixels</th>
<th>Strongly discouraged. Use only if you are certain that all users have identical screens and resolutions and will continue to have them in the future.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Characters</td>
<td>Use only if you intend to deploy to a character mode platform.</td>
</tr>
</tbody>
</table>
3.2.2.3 Adding color

- Use color sparingly, and only to get the user’s attention.
- Use color meaningfully and consistently. Example: Use color coding to differentiate between required, optional, and display-only fields, making sure that all such fields are color coded the same way.
- Consider enabling users to change the color scheme, if possible.
- Do not rely on color alone for communicating status or other information; always provide alternative cues, such as sound or other highlights. For example, if you display a negative total in red, include parentheses so that the message is clear without the use of color. Also, avoid references to specific colors in messages, as many users are color blind.
- Use object libraries or Visual Attributes to standardize color usage.
- When choosing colors, remember:
  - Red and blue combinations are hard on the eyes.
  - Blue text has a receding effect.
  - Deep blue backgrounds are hard on the eyes over long periods, as are other bright colors.
  - A significant number of people have color-identification problems, especially with red-green.
  - Colors have different implications in different countries. Follow the cultural color coding in your target market, observing the needs of different professions, situations, and so on. For example, while green generally has positive connotations for most of the western world, to those in chemical-related professions the color green might mean danger.

<table>
<thead>
<tr>
<th>Color</th>
<th>Implies...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blue</td>
<td>Cool</td>
</tr>
<tr>
<td>Black</td>
<td>Profit (financial)</td>
</tr>
<tr>
<td>Green</td>
<td>Go, OK, Danger (for chemists)</td>
</tr>
<tr>
<td>Red</td>
<td>Hot, Stop, Danger, Loss (financial)</td>
</tr>
<tr>
<td>Yellow</td>
<td>Warning, Attention</td>
</tr>
</tbody>
</table>

Table 3-4  Color Implications for the Western World
### 3.2.2.4 Creating canvases

The following table presents recommendations for creating canvases.

<table>
<thead>
<tr>
<th>Canvas Type</th>
<th>Recommendation</th>
</tr>
</thead>
<tbody>
<tr>
<td>General</td>
<td>■ Provide plenty of white space between items and regions.</td>
</tr>
<tr>
<td></td>
<td>■ Consider placing optional information on separate canvases.</td>
</tr>
<tr>
<td></td>
<td>■ Avoid scrolling windows, if possible. Studies have show that productivity decreases sharply when the user has to scroll a window to complete a task.</td>
</tr>
<tr>
<td></td>
<td>■ Plan separate windows for canvases that need to be viewed concurrently.</td>
</tr>
<tr>
<td></td>
<td>■ Although your planned layouts may fit comfortably on a monitor using Super VGA mode, they may scroll off-screen in different resolutions, like VGA. Test your layout on all your users’ monitors.</td>
</tr>
<tr>
<td>Content canvas</td>
<td>■ Set content canvases to Display immediately.</td>
</tr>
<tr>
<td></td>
<td>■ Remember that the view size for a content canvas is determined by the current size of its assigned window.</td>
</tr>
<tr>
<td></td>
<td>■ Consider using non-white canvases so that the bevel effects of objects on the canvas are maximized. In addition, white backgrounds are often so bright that they can be tiring.</td>
</tr>
<tr>
<td></td>
<td>■ Use one content canvas per window. Using more than one can be confusing if the user does not understand why the entire window is being replaced. If you do use more than one content canvas, make sure they are logically related, and require the user to move between them explicitly. One successful implementation of multiple content canvases is a word processing application in which the user chooses between several views of the same document: print preview, normal, and outline.</td>
</tr>
</tbody>
</table>

*Table 3–5 Recommendations for Creating Canvases*
3.2 Creating an Effective Form

Stacked canvas
- Use stacked canvases to hide and display groups of objects, including boilerplate.
- Size the stacked canvas only large enough to contain the necessary items.
- Be sure you know how stacked canvases behave before you implement them. For example, if the user uses Next Field or Next Record to navigate to a field that is obscured by a stacked canvas, the stacked canvas seems to disappear—that is, it is automatically placed beneath the content canvas. To re-display the stacked canvas, users must either navigate to an item on the stacked canvas or navigate away from the stacked canvas and select the Show Canvas action.

Tabbed canvas
- Limit the number of tabs to 4-6.
- Use tabs to organize related information about a single object. For example, employee information such as salary, benefits, and job description might work well as a tabbed dialog.

Table 3–5 Recommendations for Creating Canvases
### 3.2.2.5 Creating windows

The following table presents recommendations for creating windows.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Recommendations</th>
</tr>
</thead>
<tbody>
<tr>
<td>General</td>
<td>■ Do not use bevels around the edge of the window.</td>
</tr>
<tr>
<td></td>
<td>■ Inherit color settings from the environment.</td>
</tr>
<tr>
<td></td>
<td>■ Leave the top and bottom lines of the window blank, except for buttons and coordination check boxes.</td>
</tr>
<tr>
<td></td>
<td>■ Leave the left and right edge character cell columns blank, except for region lines and block boundary lines.</td>
</tr>
<tr>
<td></td>
<td>■ Use modeless (non-modal) windows to allow scrolling, and for “leave and come back” navigation (use the STD_DIALOG_WINDOW_MODELESS object in the Standard Object Library).</td>
</tr>
<tr>
<td></td>
<td>■ Use modal windows to prevent mouse navigation elsewhere and for dependent tasks that are part of a procedure (use the STD_DIALOG_WINDOW_MODAL object in the Standard Object Library).</td>
</tr>
<tr>
<td>Title</td>
<td>■ Title each window in a form uniquely so that iconified names and entries in the Windows menu are significant.</td>
</tr>
<tr>
<td>Position</td>
<td>■ Make sure each window is fully visible when it is first opened.</td>
</tr>
<tr>
<td></td>
<td>■ Make all windows moveable.</td>
</tr>
<tr>
<td></td>
<td>■ Retain window positions when a form is exited.</td>
</tr>
<tr>
<td>Scrollbar</td>
<td>■ Design your windows so that scrolling is not required by default. Scrolling is acceptable only when provided as a result of the user re-sizing the window.</td>
</tr>
<tr>
<td>Toolbar</td>
<td>■ Place the toolbar only on the container window (on Windows) or the root window (on all other platforms).</td>
</tr>
<tr>
<td></td>
<td>■ Provide hints for the toolbar buttons in tooltip help displayed just beneath each button as the mouse passes over it. (See Section 3.2.2.9.1, “Implementing Tooltips”.)</td>
</tr>
</tbody>
</table>

*Table 3–6 Recommendations for Windows*
3.2.2.5.1 Choosing a title for modeless windows

While the STD_DIALOG_WINDOW_MODELESS object in the Standard Object Library addresses all issues pertaining to positioning, closing, resizing, and placement, you still have to choose your own title. When doing so:

- If the window performs a product-specific task, use the format <Verb><Noun>, as in Transfer Items, Post Journals, and AutoCreate Quotes.
- Pluralize window names—that is, use “Items” instead of “Item”—except when the window pertains to a single instance of data.
- Provide context for child windows in the form <window title> - <context>, where context is the topmost master record or, for a new record, [New].

Examples: Assignments (OR1) - [John Doe]

Purchase Order Lines (ABC) - [New]

3.2.2.6 Creating regions

The following table presents recommendations for creating regions.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Recommendation</th>
</tr>
</thead>
<tbody>
<tr>
<td>General</td>
<td>- Avoid creating regions or adding boilerplate lines to group items unless doing so is meaningful to the user and improves the usability of the screen.</td>
</tr>
<tr>
<td></td>
<td>- Make the line or rectangle creating the region black, with an inset bevel.</td>
</tr>
<tr>
<td></td>
<td>- Use a frame for regions containing an entire block. A frame has properties that control the layout of the items within it, such as the space between the frame and items (margin), spacing, and visual attributes. Using standard frames ensures the consistency of your window layout. (Although the Layout Wizard creates a frame, you can always override it by applying a frame stored in your object library.)</td>
</tr>
</tbody>
</table>

Table 3–7 Recommendations for Regions
3.2.2.7 Adding items to blocks

The following table should help you decide when to choose one form item over another. It also presents some guidelines that you can use if you decide to modify an object or standard in the Standard Object Library. The items are presented in alphabetical order.

<table>
<thead>
<tr>
<th>Item</th>
<th>When to use</th>
<th>Recommendations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boilerplate text</td>
<td>Use for text that is neither a prompt nor a title.</td>
<td>Use mixed case.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Avoid overuse of italics and underlining.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Use font styles consistently. For example, if you use bold for emphasis, do not use bold for any other purpose.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Avoid excessive variations of fonts, sizes and colors.</td>
</tr>
</tbody>
</table>

Table 3–7 Recommendations for Regions

Table 3–8 Recommendations for Items
### 3.2 Creating an Effective Form

<table>
<thead>
<tr>
<th>Item</th>
<th>When to use</th>
<th>Recommendations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buttons</td>
<td>Use as dialog responses (in modal windows) and for item-related actions.</td>
<td>- Use one of the STD_BUTTON_TYPE objects in the Standard Object Library.</td>
</tr>
<tr>
<td>(non-iconic)</td>
<td></td>
<td>- Limit six to a window. Arrange in a single row, if possible, or a single column.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Align buttons, leaving 0.1” space between them. Separate logical groupings of buttons by 0.5”.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Leave 0.1” between the right edge of the rightmost button and the right edge of the window.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Capitalize label words; for example, ‘Print Invoice’.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Use an ellipsis (...) at the end of a button label if the button opens a modal window or if the user must provide more information about the action in another window (modal or not) before the action can be completed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Place OK and Cancel buttons together.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Put affirmative and cancellation buttons first, unique buttons last.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Use chevrons (&gt;&gt;) to indicate that the dialog will be expanded.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- For labelled buttons (except OK and Cancel), always provide an accelerator key (underlined letter).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Use the first letter of the first or second word in the label (&quot;F&quot; for “File” or “P” for “Start Posting”). If a stronger link exists (like “X” for “Exit”), use that letter.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Use consonants instead of vowels when possible.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Make access keys unique within a window. Ensure they do not conflict with the keys used by the top level of the menu.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Check boxes</th>
<th>Use only when the label on the check box can clearly be thought of as having “true” (checked) and “false” (unchecked) states. Otherwise, use a radio button group with two items.</th>
<th>- Use the STD_CHECKBOX object in the Standard Object Library.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>- Use positive statement labels:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Not good: Don’t show this alert in the future.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Better: Show this alert in the future.</td>
</tr>
</tbody>
</table>

*Table 3-8  Recommendations for Items*
### 3.2 Creating an Effective Form

#### Display items
- Use for display-only fields in which the user can never type; for example, the Total field in a financial application.
- Use the `STD_DISPLAY_ITEM` object in the Standard Object Library.

#### Icons
- Use only for frequent or critical actions.
- Use where a picture conveniently conveys a task or mimics a real-world object.
- Place frequently used buttons on a toolbar.
- Group related tools together and separate groups with white space.
- Disable buttons that are unavailable.
- Always provide tooltips, as users are often confused by the meaning of icons.
- Icons are often cultural. Be aware that you may need to translate them.
- Avoid “visual puns”, such as a running figure for “Run”. Their meanings are not obvious, and will certainly not be understood in other languages.

#### Lists (see also Poplists and T-Lists)
- Use when it is easier for the user to select a value than to type in a value.
- Use for data entry and display of text values in a selectable list format.
- Use when displayed value entries are relatively short (up to 30 characters each).
- Use the combo-box style if the user may enter new values.
- For 15 entries or less, use a poplist. For more than 15 entries, use an LOV (List of Values). For more than 30 entries with a lot of real estate, use a T-List.
- Make all related fields the same length.
- Use the color of the canvas background for text items that have become non-enterable.

#### LOVs
- Use when the user must select from a list of more than 15 rows or to show several columns of data.
- Automatically select a row for the user when there is only one valid value.
- Move the cursor automatically to the next field after a selection is made.
- If there are more than 100 rows in the LOV, prompt the user to reduce the list of valid values before making a selection.

#### Poplists
- Use when only one value is applicable and the list of choices is 15 or less.
- Before implementing a poplist, consider whether frequent users can type faster than they can select.

<table>
<thead>
<tr>
<th>Item</th>
<th>When to use</th>
<th>Recommendations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display items</td>
<td>Use for display-only fields in which the user can never type; for example, the Total field in a financial application.</td>
<td>Use the <code>STD_DISPLAY_ITEM</code> object in the Standard Object Library.</td>
</tr>
<tr>
<td>Icons</td>
<td>Use only for frequent or critical actions.</td>
<td>Place frequently used buttons on a toolbar.</td>
</tr>
<tr>
<td></td>
<td>Use where a picture conveniently conveys a task or mimics a real-world object.</td>
<td>Group related tools together and separate groups with white space.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Disable buttons that are unavailable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Always provide tooltips, as users are often confused by the meaning of icons.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Icons are often cultural. Be aware that you may need to translate them.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Avoid “visual puns”, such as a running figure for “Run”. Their meanings are not obvious, and will certainly not be understood in other languages.</td>
</tr>
<tr>
<td>Lists (see also Poplists and T-Lists)</td>
<td>Use when it is easier for the user to select a value than to type in a value.</td>
<td>For 15 entries or less, use a poplist. For more than 15 entries, use an LOV (List of Values). For more than 30 entries with a lot of real estate, use a T-List.</td>
</tr>
<tr>
<td></td>
<td>Use for data entry and display of text values in a selectable list format.</td>
<td>Make all related fields the same length.</td>
</tr>
<tr>
<td></td>
<td>Use when displayed value entries are relatively short (up to 30 characters each).</td>
<td>Use the color of the canvas background for text items that have become non-enterable.</td>
</tr>
<tr>
<td></td>
<td>Use the combo-box style if the user may enter new values.</td>
<td></td>
</tr>
<tr>
<td>LOVs</td>
<td>Use when the user must select from a list of more than 15 rows or to show several columns of data.</td>
<td>Automatically select a row for the user when there is only one valid value.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Move the cursor automatically to the next field after a selection is made.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If there are more than 100 rows in the LOV, prompt the user to reduce the list of valid values before making a selection.</td>
</tr>
<tr>
<td>Poplists</td>
<td>Use when only one value is applicable and the list of choices is 15 or less.</td>
<td>Before implementing a poplist, consider whether frequent users can type faster than they can select.</td>
</tr>
</tbody>
</table>
### 3.2 Creating an Effective Form

<table>
<thead>
<tr>
<th>Item</th>
<th>When to use</th>
<th>Recommendations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pop-up menus</td>
<td>Use to associate menu options with an item, rather than the whole application.</td>
<td>Use the <code>STD_POPUP_MENU_ITEM</code> object in the Standard Object Library.</td>
</tr>
<tr>
<td></td>
<td>Use to provide access to frequently used commands.</td>
<td></td>
</tr>
<tr>
<td>Prompts</td>
<td>Use as labels for fields, check boxes, lists, etc.</td>
<td>Place toward the top or left of the element they are describing.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Always place single-record blocks prompts to the left of the field and multi-record block prompts above the field.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Use the terms “From” and “To” to identify fields involved in a range rather than “Start” and “End” or “Low” and “High”.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For percentages, place the percent sign (%) after the field. Do not include it in the prompt.</td>
</tr>
<tr>
<td>Radio groups</td>
<td>Use to present mutually exclusive choices.</td>
<td>Use the <code>STD_RADIO_GROUP</code> object in the Standard Object Library.</td>
</tr>
<tr>
<td></td>
<td>Use to set a ‘mode’, such as what type of information will be displayed.</td>
<td>Use vertical orientation instead of horizontal.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Group related buttons into radio groups with a title.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If the choices are binary (ON/OFF, YES/NO), use a check box instead.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Always provide a default value.</td>
</tr>
<tr>
<td>T-Lists</td>
<td>Use only when one value is applicable and the list of choices is never expected to grow beyond 30.</td>
<td>Always show at least five rows of data.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Use only in forms with a lot of available real estate.</td>
</tr>
<tr>
<td>Text items</td>
<td>Use for data entry and display of character values.</td>
<td>Use the color of the canvas background for text items that have become non-enterable.</td>
</tr>
<tr>
<td></td>
<td>Use for lengthy or unprepared values (that is, those that do not appear in a short, pre-defined list).</td>
<td>Use a bevel if the user can enter values in the field.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Use the <code>STD_TEXT_ITEM</code> or one of the <code>STD_DATE_TYPE</code> objects in the Standard Object Library.</td>
</tr>
</tbody>
</table>

**Table 3–8 Recommendations for Items**
3.2.2.8 Designing messages

Messages are shown either in the window console area or in popup windows called alerts. How you display messages depends upon their type and whether a reply is required by the user. Here are some suggestions:

<table>
<thead>
<tr>
<th>Message Type</th>
<th>Recommendations</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Errors</td>
<td>To present an error message, use the STD_ALERT_STOP object under the Alerts tab in the Standard Object Library.</td>
<td>“You do not have sufficient authority to approve this Order.”</td>
</tr>
<tr>
<td></td>
<td>Use when an error is serious enough to halt processing. Include a stop sign icon in the dialog.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Use sparingly.</td>
<td></td>
</tr>
<tr>
<td>Warning</td>
<td>Use the STD_ALERT_CAUTION_1, STD_ALERT_CAUTION_2, or STD_ALERT_CAUTION_3 objects under the Alerts tab in the Standard Object Library. While in the Library, click the object once to see a description of the message text.</td>
<td>“Copy all lines on this invoice?”</td>
</tr>
<tr>
<td></td>
<td>Use to present a question that the user must respond to before processing continues. Include a yield sign icon (!) in the dialog.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Keep the warning short and concise. For example, use “Delete this order?” rather than “Do you really want to delete this order?”</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Phrase questions positively (“Save changes?” rather than “Are you sure you don’t want to save changes?”)</td>
<td></td>
</tr>
<tr>
<td>Information</td>
<td>Use the STD_ALERT_INFORMATION object in the Standard Object Library.</td>
<td>“Line and Shipment Quantities currently do not match.”</td>
</tr>
<tr>
<td></td>
<td>Use to present messages that the user must acknowledge when no choice is involved. Include the information icon (the letter “i” in a circle) and the OK button.</td>
<td>“There are items awaiting your attention.”</td>
</tr>
<tr>
<td>Hints</td>
<td>Appears on the Form Builder message line in the Console.</td>
<td>“Working...”</td>
</tr>
<tr>
<td></td>
<td>Use to present messages of very little consequence, or process indicators that do not require a response.</td>
<td>“At first record.”</td>
</tr>
<tr>
<td></td>
<td></td>
<td>“Processed Order line 12 of 37.”</td>
</tr>
</tbody>
</table>

Table 3–9 Displaying messages
3.2.2.8.1 Creating Message Text

If possible, error messages should include:

- What was done
- Cause (why it was wrong)
- Action (how to fix it)

Here are some examples of bad and good message text:

<table>
<thead>
<tr>
<th>Bad</th>
<th>Good</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invalid Date</td>
<td>“Please re-enter the date as DD-MON-YY.”</td>
</tr>
<tr>
<td>Do not enter a start date</td>
<td>“The start date must be earlier than the end date.”</td>
</tr>
<tr>
<td>later than the ending date</td>
<td></td>
</tr>
<tr>
<td>Error: 1623, constraint violation</td>
<td>“Please re-enter a unique value for this field.”</td>
</tr>
<tr>
<td>You should not receive this message</td>
<td>Don’t display the message at all.</td>
</tr>
<tr>
<td>Tool lost at sea</td>
<td>Replace it with an appropriate message or take it out altogether</td>
</tr>
</tbody>
</table>

When writing message text, try to adhere to these guidelines:

<table>
<thead>
<tr>
<th>Recommendation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use active voice.</td>
<td>“Do this now”, not “This will need to be done”</td>
</tr>
<tr>
<td>Use imperative voice.</td>
<td>“Enter a commission plan”, not “You can enter a commission plan”</td>
</tr>
<tr>
<td>Use “can” instead of “may” or “could”.</td>
<td>“You cannot delete a printed release”, not “You may not delete a printed release”</td>
</tr>
<tr>
<td>Refer to actual field names when possible.</td>
<td>If a field is labelled “Sales Associate”, don’t use the message “Please enter a different salesperson”.</td>
</tr>
<tr>
<td>Use uppercase for commands and keywords.</td>
<td>ALTER CLUSTER statement is no longer supported.”</td>
</tr>
<tr>
<td>Avoid the use of humor.</td>
<td>You made a boo-boo!</td>
</tr>
</tbody>
</table>

Table 3–10 Recommendations for Message Text
3.2 Creating an Effective Form

3.2.2.9 Implementing online help

This section discusses using two types of online help:

- **Toolips**: Also known as popup hints and microhelp. Displayed when the user moves the mouse over an item on the screen.
- **Online Help**: Contains context-sensitive help and hypertext links that enable users to jump to related topics.

### 3.2.2.9.1 Implementing Toolips

Each item has a property called Toolip and another called Toolip Visual Attribute Group. In the Property Palette’s Toolip property field, enter the text you want to display in the pop-up. To ensure consistency across your application, apply the **STD_TOOLTIP** visual attribute from the Standard Object Library. If you don’t apply a visual attribute, the toolip uses a platform-specific default.

### 3.2.2.9.2 Implementing Online Help

- Consider using help authoring tools.
- Create standalone topics that are hyperlinked to other topics.
- Keep text short and concise.
- Provide a Help button on each dialog, a Help option on the main menu, and a Help button on the toolbar.

---

### Table 3–10 Recommendations for Message Text

<table>
<thead>
<tr>
<th>Recommendation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avoid accusatory messages. Do not insinuate that the user is at fault. Do not mention the user’s mistake unless it pertains to the problem’s solution.</td>
<td>Instead of “You didn’t choose a value”, try “Please choose a value”.</td>
</tr>
<tr>
<td>When a message contains instructions, use “please”.</td>
<td>“Please choose a value” is preferred over “Choose a value”.</td>
</tr>
<tr>
<td>Address the user as “you”, not “the user”. Avoid using “I”, “He”, or “She”.</td>
<td>Instead of “The user should back up his modules”, try “Please back up your modules”.</td>
</tr>
<tr>
<td>Consider providing context-sensitive help when errors occur.</td>
<td>See Section 3.2.2.9.2, “Implementing Online Help”.</td>
</tr>
</tbody>
</table>

---

Designing Visually Effective Applications 3-187
3.2.2.10 Building effective menus
Form Builder provides a default menu for every form. The default menu includes commands for all basic database operations, including querying, inserting, and deleting. If your application has specific requirements not met by the default menu you can quickly create a custom menu. (See “Creating a menu” in the Form Builder online help for instructions.) While building a menu, keep the following ideas in mind:

- Organize commands according to the tasks they belong to.
- Although Developer/2000 supports scrolling menubars, try to keep valid items on the screen where the user can see them.
- Disable items that are unavailable.
- Limit submenus to two levels, if possible.
- Capitalize labels.
- Use the standard menus in the demos and add-ons as models for your own menus.

3.3 Creating an Effective Report
The first steps in using Report Builder to design an effective report are the same as those for designing an effective form or display. Before reading the rest of this section, it’s a good idea to read Section 3.1.2, “Defining user requirements”, if you haven’t already.

Here are a few questions to help you determine the user requirements for your report:

- What data will people viewing the report want, and in what priority?
- Will users want to “drill down” on data, so they can see more details? If so, you’ll want to include buttons in your reports. Buttons can have blocks of PL/SQL code associated with them, so they can invoke secondary reports, play videos or sounds, and so on.
- Will users want charts in the report to present data visually? If so, what data? You can create a chart in Graphics Builder, then pass data to the chart from Report Builder (instead of performing a second query).
- If users modify data using a form application, will they want to print the data afterward? If so, you’ll want to call the report from a form, and have the form pass data to the report.
3.3 Creating an Effective Report

- Will users want a report to be embedded in a form? If so, you’ll want to design a template that has font and color standards similar to your forms.

- Will users want to view the report in HTML, PDF, or hardcopy? Will they want to make a few formatting changes in the Live Previewer before (optionally) printing? If so, you will need to specify the report destination parameter (DESTYPE), or enable your users to do so.

- Will users want to specify parameters for a report, as in “Show only the top 10 sales for the userid STIGER”? If so, you will need to create user parameters, and have users specify their values in a form or the Runtime Parameter dialog.

- Based on network traffic and machine performance, should the report run in a client/server or 3-tiered architecture?

- Do you have a corporate standard that you want to propagate in the reports? If so, you should define standard templates.

- For Web reports, will the number of reports be static, or do you want to dynamically generate the Web sites?

3.3.1 Understanding Reports

Before addressing specific considerations for reports, it may be helpful to briefly introduce some basic reports concepts. (Experienced users of Report Builder should skip this section.) For more details on these and other related reports topics, see the Report Builder online help and/or the Report Builder section of the Developer/2000 Quick Tour.

When you build a report, you work with two application components:

<table>
<thead>
<tr>
<th>Module Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Report module</td>
<td>A collection of objects and code routines. Some of the objects you can define in a report module include repeating frames, frames, fields, boilerplate, anchors, and blocks of PL/SQL code called triggers.</td>
</tr>
<tr>
<td>PL/SQL Library module</td>
<td>A collection of user-named procedures, functions, and packages that can be called from other modules in the application.</td>
</tr>
</tbody>
</table>

Table 3–11 Report Builder Modules

This section does not address the use of PL/SQL libraries. For information on this topic, refer to the Report Builder online help.
For some reports, you will use the Report Wizard (to choose a report type, define a
data model, and a layout for the data) and the Report Editor’s Live Previewer (to
fine-tune the report). For other reports, you will use other views of the Report Edi-
tor:

<table>
<thead>
<tr>
<th>View</th>
<th>Used to:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Model view</td>
<td>Create a report with more than one query.</td>
</tr>
<tr>
<td>Layout Model view</td>
<td>■ Create reports with multiple sections (e.g., a single report</td>
</tr>
<tr>
<td></td>
<td>with a tabular and matrix style)</td>
</tr>
<tr>
<td></td>
<td>■ Add new layout objects (e.g., buttons)</td>
</tr>
<tr>
<td></td>
<td>■ Control how objects are sized or positioned</td>
</tr>
<tr>
<td>Parameter Form view</td>
<td>Present users with a dialog in which they can specify parameter</td>
</tr>
<tr>
<td></td>
<td>values before running the report.</td>
</tr>
</tbody>
</table>

*Table 3–12  Report Editor Views*

Because this chapter discusses how to create visually effective applications, the
remainder of this section focuses on using the templates, objects, and settings
found in the Layout Model view.

### 3.3.2 Using Templates in Report Builder

Perhaps the most important means of standardization available to you as a report
developer is the *template*. A template is a collection of boilerplate and layout set-
tings that determine the appearance of an entire report. Several templates are
shipped with Report Builder, and you can create your own. By creating corporate
or group templates and making them available to your entire development team,
you can ensure a common look and feel. For instructions on how to create a tem-
plate, see the Report Builder online help.
3.3.3 Understanding Layout Objects

The Layout view of the Report Editor may contain the following objects:

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frames</td>
<td>Containers that control repeating frames, fields, boilerplate, buttons, and child frames. Unlike Form Builder frames, Report Builder frames do not have formatting properties that control the location of child objects.</td>
</tr>
<tr>
<td>Repeating frames</td>
<td>Containers that control:</td>
</tr>
<tr>
<td></td>
<td>- Fields containing report data</td>
</tr>
<tr>
<td></td>
<td>- Other objects owned by the repeating frame</td>
</tr>
<tr>
<td>Fields</td>
<td>Containers that display the report data, dates, page numbers, and so on.</td>
</tr>
<tr>
<td>Boilerplate</td>
<td>Text or graphics that appear as often as required by the object that surrounds it (the report, frame, or repeating frame), or to which it is attached.</td>
</tr>
<tr>
<td>Anchors</td>
<td>Objects that determine how two objects in a report layout relate to one another (i.e., parent/child relationships and relative positioning).</td>
</tr>
<tr>
<td>Buttons</td>
<td>Objects that perform an action when users clicks on them.</td>
</tr>
</tbody>
</table>

*Table 3–13  Layout Model Objects*

With the exception of anchors, layout objects may have format triggers, such as PL/SQL blocks that are invoked each time the object is activated.

3.3.4 Controlling Layout Objects in Report Builder

When designing a report, remember that the size of the entire report and the size of many of its individual objects may vary, which can affect pagination in a printed report. Consider a report based on this query:

```sql
select ename, sal from emp
where sal > 2000
```

The size of this report is based on several factors:

- The amount of data that satisfies the query, which can range from a few records, to hundreds, to thousands.
3.3 Creating an Effective Report

- When you run the report. For example, the number of records might change dramatically the week after pay raises are distributed (for example, the number of people with salaries above 2000 might increase).

- Whether group filters or format triggers are used, which exclude data or objects from the report.

Instances of the same object may also vary in size. For example, suppose you have a VARCHAR2 column in the database called COMMENTS. For one record, COMMENTS might contain two sentences. For another, it might contain 10 sentences. The size of the field in your layout that corresponds to the COMMENTS column must then be able to accommodate values of different length. In addition, objects around that field may have to be “pushed” or “pulled” to avoid being overwritten or leaving large gaps in the report.

Fortunately, Report Builder provides a variety of mechanisms in the Layout View of the Report Editor that enable you to control how objects are sized and positioned. These mechanisms are described in the following sections:

- Section 3.3.4.1, “Using anchors”
- Section 3.3.4.2, “Using the Print Object On and Base Printing On properties”
- Section 3.3.4.3, “Understanding Horizontal and Vertical Elasticity”
- Section 3.3.4.4, “Using the Page Break Before and After property”
- Section 3.3.4.5, “Using the Page Protect property”
- Section 3.3.4.6, “Using the Keep with Anchoring Object property”

3.3.4.1 Using anchors

Anchors determine how objects in a report layout relate to one another. When two objects are anchored together, one object is considered the parent and the other the child. By defining parent-child relationships between objects, anchors establish a hierarchy for the objects in a report. Based on this hierarchy of objects, Report Builder decides how objects should be printed in relation to each other, whether it should attempt to keep the two objects on the same page, and how objects should be pushed or pulled depending on the size of surrounding objects.

Anchors can be created in one of two ways:

- Automatically, by Report Builder. This is known as an implicit anchor. In most cases, implicit anchors are the only ones you need.

- By you, using the Anchor Tool in the Layout view of the Report Editor. This is known as an explicit anchor. Explicit anchors are necessary only when you
need to override the implicit anchors for some reason. See “Creating anchors” in the Report Builder online help.

### 3.3.4.2 Using the Print Object On and Base Printing On properties
The Print Object On property determines the frequency with which an object appears in a report. The Base Printing On property specifies the object on which to base the Print Object On property.

For example, if you specify a Print Object On of All Pages and a Base Printing On of Anchoring Object, the object is triggered to print on every logical page on which its anchoring object (parent object) appears. Objects created by the Report Wizard have these properties set for them. In most cases, the values that Report Builder chooses are the best ones for the object. The only time you should need to set these properties yourself is when you want to override the default value set by Report Builder.

In applying the Print Object On property, Report Builder considers the first page of an object to be the first logical page on which some part of the object is printed. Likewise, the last page is considered to be the last logical page on which some part of the object is printed. For example, if you specify a Print Object On of First Page and a Base Printing On of Enclosing Object, the object will be triggered to print on the first logical page on which its enclosing object appears.

**Notes:**

- If an object is inside a repeating frame, Base Printing On refers to each instance of the repeating frame. If the object is outside the repeating frame and explicitly anchored to it, Base Printing On refers to the repeating frame as a whole.
- Just because an object is triggered to print on a logical page does not mean it will print on that logical page. Other settings (e.g., Page Break Before) or the amount of space available on the page may cause Report Builder to print an object on a page other than the one on which it was initially triggered to print.

For more information, refer to the Report Builder online help, index entries: Print Object On and Base Printing On.

### 3.3.4.3 Understanding Horizontal and Vertical Elasticity

The Horizontal and Vertical Elasticity properties determine how the horizontal and vertical sizes of the object may change at runtime to accommodate the objects or data within it:

- For frames and repeating frames, elasticity defines whether the size of the frame or repeating frame should vary with the objects inside of it.
For objects containing text, elasticity defines whether the field or boilerplate should vary with the size of the text. Fixed size text will wrap within the defined size of the object and may be truncated if there is not enough room. Number or date data will appear as asterisks if the data cannot fit within the defined size.

For images, drawings, and chart objects, Report Builder uses proportional scaling. The elasticity options for images, drawings, and chart objects determine the scaling factor.

Objects created by the Report Wizard have these properties set for them. In most cases, the values that Report Builder chooses are the best ones for the object. The only time you should need to set these properties yourself is when you want to override the default value set by Report Builder.

Different elasticity settings can produce unexpected results in the output. For example, if an object with variable horizontal elasticity contracts, all objects to the right are moved to the left, since these objects are implicitly anchored to the variable object. If, however, this does not occur when the object’s horizontal elasticity is set to contract, since objects are not implicitly anchored in this case.

For more information, refer to the Report Builder online help, index entries: Horizontal Elasticity and Vertical Elasticity.

3.3.4.4 Using the Page Break Before and After property

Unlike word processing documents, reports and their objects can vary in size and position at runtime. As a result, page breaks in a report can be difficult to predict.

Use the Page Break Before property to indicate that you want an object to be formatted on the page after the page on which it is initially triggered to print. Note that this does not necessarily mean that all the objects below the object with Page Break Before will move to the next page. If one of the objects below does not have Page Break Before set and can fit on the page, it may print above the object which has Page Break Before set.

Use the Page Break After property to indicate that you want all children of the object to move to the next page. In other words, any object that is a child object of an anchor (implicit or explicit) to this object will be treated as if it has Page Break Before set. Note that this does not necessarily mean that all the objects below the object with Page Break After will move to the next page. If one of the objects below does not have Page Break After set and is not a child of the other object, it might print above the object which has Page Break After set.
3.4 Creating an Effective Display

Graphics Builder enables you to produce displays for inclusion in both forms and reports. A display can be an application by itself, or included in a form or report.

Use displays when you want to:

- Show relationships between different categories (number of tennis shoes sold as compared to dress shoes)
- Show trends rather than specific values
- Provide user interaction with graphical areas and shapes (maps, sectors of images, and so on). Graphics Builder allows you to respond to mouse interactions with the shapes that you create in the layout editor. Irregular, transparent
buttons can be placed over areas of a diagram or bitmapped image so that users can effectively make selections from pictures.

When creating graphics, keep the following guidelines in mind:

- Keep things simple. Displays containing too many lines, bars, slices, and so on can quickly overwhelm users and render your graph or chart unusable. If you have a lot of data, summarize it at the highest level and use drill-downs to present more detailed information. Or, consider breaking up a complicated graph into smaller, individual graphs, then creating a form from which users can select which graph they want to view.
- Use 3-D effects only if they help communicate information, as they can be resource-intensive.
- Test your graphics on all the display devices in your deployment environment and make sure they perform well even on the lowest resolution monitor.
- Use colors to show transition; use primary colors to show differences. See Section 3.2.2.3, “Adding color” for more information on using color.
- Use legends for complicated graphs.
- Remember, you can pass mouse events from forms to graphics modules. For example, you can create a When-Mouse-Click trigger on a form’s chart item and call the OG.MouseDown procedure from this trigger to pass mouse information to a display. The display can then return information to the form, including details of which sector in the display was clicked by the user. See the Graphics Builder demo called “Map Example” in the product’s standard demo set for more information.

### 3.4.0.7 Choosing the Right Graph
Here are some guidelines for implementing the most commonly used displays:

<table>
<thead>
<tr>
<th>Display type</th>
<th>When to use:</th>
<th>Recommendations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bar graph</td>
<td>Showing relationships between discrete objects and their related values.</td>
<td>Limit bars to 20-25.</td>
</tr>
<tr>
<td>Pie chart</td>
<td>Showing part-to-whole relationships. Typically used to show percentage values.</td>
<td>Limit slices to 10.</td>
</tr>
</tbody>
</table>

*Table 3–14  Recommendations for Displays*
<table>
<thead>
<tr>
<th>Display type</th>
<th>When to use:</th>
<th>Recommendations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Line chart</td>
<td>Show the cumulative effect of continuous data.</td>
<td>Limit lines to 6-8.</td>
</tr>
<tr>
<td>Double-Y</td>
<td>Comparing data within a large range of values.</td>
<td>Limit plots to 4 or less.</td>
</tr>
<tr>
<td>Gantt</td>
<td>Scheduling and date duration data.</td>
<td>Limit bars to 40-50.</td>
</tr>
<tr>
<td>High-low</td>
<td>Displaying daily temperature values, stock market values, and similar data tracking high, low, and current values.</td>
<td>Limit rows to 30 or less.</td>
</tr>
<tr>
<td>Mixed</td>
<td>Comparing actual values (bar) to projected values (line).</td>
<td>Limit rows to 30 or less.</td>
</tr>
<tr>
<td>Scatter</td>
<td>Showing relationships between numeric data on the X and Y axis.</td>
<td>Limit rows to less than 50 per inch.</td>
</tr>
</tbody>
</table>

*Table 3–14  Recommendations for Displays*
3.4 Creating an Effective Display
This chapter explains how to design multilingual applications with Developer/2000.

4.1 National Language Support (NLS)

Oracle’s National Language Support makes it possible to design multilingual applications. A multilingual application is an application which can be deployed in several different languages. Oracle supports most European, Middle Eastern, and Asian languages.

National Language Support makes it possible to:

- use international character sets (including multibyte character sets).
- display data according to the appropriate language and territory conventions.
- extract strings that appear in your application’s user interface and translate them.

4.1.1 The language environment variables

Three language environment variables are available: NLS_LANG, DEVELOPER_NLS_LANG, and USER_NLS_LANG.

4.1.1.1 NLS_LANG

The NLS_LANG language environment variable specifies the language for messages displayed to the user, such as the “Working...” message. NLS_LANG also specifies the default format masks used for DATE and NUMBER datatypes, the sorting sequence, and the character set.
The syntax for NLS_LANG is as follows:

\[\text{NLS_LANG} = \text{language}_{-}\text{territory}.\text{char\_set}\]

- **language** specifies the language and its conventions for displaying messages and day and month names.
- **territory** specifies the territory and its conventions for default date format, decimal character used for numbers, currency symbol, and calculation of week and day numbers.
- **char\_set** specifies the character set in which data is displayed.

For example, let’s say you want your application to run in French. The application will be used in France. Data will be displayed using the WE8DEC character set. You would set NLS_LANG as follows:

NLS_LANG=French_France.WE8DEC

Now let’s say you want your application to run in French, but this time the application will be used in Switzerland. You would set NLS_LANG as follows:

NLS_LANG=French_Switzerland.WE8DEC

**Note:** You are strongly advised to set the language and territory parameters of NLS_LANG to the same values on the server side and the client side. (The value of the char_set parameter on the server side is specified when the database is created and cannot be changed.) Use the SQL command ALTER SESSION to change the values of the language and territory parameters on the server side. For example, this statement changes the language parameter to French and the territory parameter to France:

```
ALTER SESSION
  SET NLS_LANGUAGE = French NLS_TERRITORY = France
```

### 4.1.1.2 DEVELOPER_NLS_LANG and USER_NLS_LANG

If you must use two sets of resource and message files at the same time, two other language environment variables are available: DEVELOPER_NLS_LANG and USER_NLS_LANG. DEVELOPER_NLS_LANG and USER_NLS_LANG take the same parameters as NLS_LANG. Use these variables instead of NLS_LANG in the following situations:

- If you are a developer who prefers to use the Designer in English, but you are developing an application for another language, the two variables allow you to use different language settings for the Designer and Runtime.
- If you are a developer creating an application to run in a language for which a local-language version of the Designer is not currently available.
If these environment variables are not specifically set, they take their default values from NLS_LANG.

### 4.1.1.3 Form Builder NLS parameters

You can use Form Builder built-in functions to obtain the current value of the language environment variables for use in trigger code:

<table>
<thead>
<tr>
<th>Environment Variables</th>
<th>DEVELOPER_NLS_LANG (defaults to NLS_LANG)</th>
<th>USER_NLS_LANG (defaults to NLS_LANG)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Built-in Parameter</td>
<td>GET_FORM_PROPERTY</td>
<td>GET_APPLICATION_PROPERTY</td>
</tr>
<tr>
<td></td>
<td>MODULE_NLS_LANG</td>
<td>USER_NLS_LANG</td>
</tr>
</tbody>
</table>

Because both USER_NLS_LANG and DEVELOPER_NLS_LANG default to the value of NLS_LANG, the Form Builder NLS parameters will hold the value of NLS_LANG if either variable is not specifically set.

Both Form Builder NLS parameters have four variations which allow you to retrieve either the complete environment variable or a specific portion of it. This table shows the four parameters of the GET_APPLICATION_PROPERTY built-in that return the USER_NLS_LANG environment variable:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>USER_NLS_LANG</td>
<td>Entire USER_NLS_LANG variable</td>
</tr>
<tr>
<td>USER_NLS_LANGUAGE</td>
<td>Language portion only</td>
</tr>
<tr>
<td>USER_NLS_TERRITORY</td>
<td>Territory portion only</td>
</tr>
<tr>
<td>USER_NLS_CHARACTER_SET</td>
<td>Character set portion only</td>
</tr>
</tbody>
</table>

To retrieve the DEVELOPER_NLS_LANG environment variable, call GET_FORM_PROPERTY using the MODULE_NLS_LANG parameter.

Refer to Section 4.1.1, “The language environment variables” for more information on the language environment variables.

### 4.2 Language and Territory Default Format Masks

The language environment variable specifies the set of default format masks used to display data such as day and month names, numbers, dates, and currency. Specifi-
Form Builder uses the default format masks associated with the territory specified in the current language environment variable:

- in the Designer: When Form Builder displays default values for items, ranges, or parameters.
- at runtime: If a user enters data in a text item whose type is territory-specific, such as DATE or NUMBER.

For example, suppose that you create an item of type DATE in the Designer and then enter a default value of 20-DEC-97 using the NLS_LANG default of American_America. If you then change NLS_LANG to Norwegian_Norway, the default value for the item changes to 20.12.1997.

Refer to Section 4.1.1, “The language environment variables” for more information on the language environment variables.

### 4.2.1 Using ALTER SESSION to change the default format masks

You can use the SQL command `ALTER SESSION` in conjunction with the number format elements to change the default format masks specified by the language environment variable.

<table>
<thead>
<tr>
<th>Element</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>ISO currency symbol</td>
</tr>
<tr>
<td>L</td>
<td>Local currency symbol</td>
</tr>
<tr>
<td>D</td>
<td>Decimal separator</td>
</tr>
<tr>
<td>G</td>
<td>Group (thousands) separator</td>
</tr>
</tbody>
</table>

For example, this statement changes the decimal character to a comma and the group separator to a period:

```sql
ALTER SESSION

SET NLS_NUMERIC_CHARACTERS = ‘,.’
```
These new characters are returned when you use their number format elements D and G:

```sql
SELECT TO_CHAR(SUM(sal), 'L999G999D99') Total FROM emp
```

```
TOTAL
-------------------
FF29.025,00
```

This statement changes the ISO currency symbol to the ISO currency symbol for the territory America:

```sql
ALTER SESSION
    SET NLS_ISO_CURRENCY = America
```

The ISO currency symbol defined for America is used:

```sql
SELECT TO_CHAR(SUM(sal), 'C999G999D99') Total FROM emp
```

```
TOTAL
-------------------
USD29,025.00
```

This statement changes the local currency symbol to DM:

```sql
ALTER SESSION
    SET NLS_CURRENCY = 'DM'
```

The new local currency symbol is returned when you use the L number format element:

```sql
SELECT TO_CHAR(SUM(sal), 'L999G999D99') Total FROM emp
```

```
TOTAL
-------------------
DM29.025,00
```

Refer to Section 4.1.1, “The language environment variables” for more information on the language environment variables.
4.2.2 Format mask design considerations

When working with date and currency fields in multilingual applications, you should make all screen items (boilerplate, text items, interface objects such as buttons and lists of values) longer to allow for translation of text and different ways of displaying data. For example, if you develop an application in American English with a 9-character DD-MON-YY date and then run the application in Norwegian, you must increase the size of the field to allow for the 10-character Norwegian date DD.MM.YYYY.

You should also consider whether you need to use the format mask characters to create special format masks or if the default format masks specified by the territory component of NLS_LANG are acceptable.

For implicit datatype conversions, PL/SQL always expects items in the American_America default format DD-MON-YY, so if you use an item whose type is territory-specific in PL/SQL, you must specify the correct format masks. Use TO_DATE to translate territory-specific items in PL/SQL.

Avoid hard-coding a string containing a month name. If a hard-coded month name is essential, avoid using the COPY built-in. If you use COPY, the month name may be incorrect, depending on which language is specified.

Language-dependent example (not recommended):

:emp.hiredate := '30-DEC-97';
copy ('30-DEC-97','emp.hiredate');

Language-independent example (recommended):

:emp.hiredate := TO_DATE('30-12-1997','DD-MM-YYYY');

Refer to Section 4.1.1, “The language environment variables” for more information on the language environment variables.

4.3 Character Sets

The character set component of the language environment variable specifies the character set in which data is represented in the user’s environment. SQL*Net ensures that data created using one character set can be correctly processed and displayed on a system that uses a different character set, even though some characters may be represented by different binary values in the different character sets.

Refer to Section 4.1.1, “The language environment variables” for more information on the language environment variables.
4.3 Character Sets

4.3.1 Character set design considerations

If you are designing a multilingual application, or even a single-language application that will run with multiple character sets, you should determine the character set most widely used at runtime and generate with the language environment variable set to that character set.

If you design and generate an application in one character set and run it in another character set, performance may suffer. Furthermore, if the runtime character set does not contain all the characters in the generate character set, question marks will appear in place of the unrecognized characters.

PDF does not support multibyte character sets.

4.3.2 Unicode

Unicode (UCS-2) is a fixed-length two-byte character set which can represent up to 65,536 characters, including Latin, Extended European, Cyrillic, Greek, Arabic, Chinese, Japanese, Korean, technical symbols, and special characters used in publishing. Currently, Windows NT 4.0 is the only operating system which fully supports Unicode (UCS-2).

Two transformations of UCS-2 have been developed to accommodate 7-bit data transmission and 8-bit UNIX environments: UTF-7 and UTF-8. UTF-8 is a variable-length character set of up to four bytes.

4.3.2.1 Unicode support in Developer/2000 Release 2.0

Developer/2000 2.0 on Windows NT 3.51 and 4.0 provides Unicode support; Windows NT 4.0 is the preferred platform. If you use Unicode, you will be able to display multiple languages in one application, although performance will suffer for single-language applications.

The Oracle database (7.3 and above) supports UTF-8 data storage. Developer/2000 2.0 can retrieve UTF-8 data from the database and store UTF-8 data to the database. (Developer/2000 2.0 uses UCS-2 internally.)

4.3.2.1.1 Font support

Developer/2000 relies on the Windows operating system for the font and input method for different languages. To enter and display text in a particular language, you must be running a version of Windows that supports that language. Font support is limited but not restricted to the Windows NT operating system font support.

Windows NT 4.0 provides True Type Big Fonts. These fonts contain all the characters necessary to display or print multilingual text. If you try to type, display, or
print multilingual text and see unexpected characters, you are probably not using a Big Font. Big Fonts provided by Microsoft under NT 4.0 are as follows: Arial, Courier New, Lucida Console, Lucida Sans Unicode, and Times New Roman. Third-party Unicode fonts are also available.

**Note:** The fixed system font (Fixedsys) will not work for strings containing non-ASCII characters. It is not a Big Font and thus will support only one character set.

4.3.2.1.2 Enabling Unicode support To enable Unicode support in Developer/2000, set NLS_LANG as follows:

```
NLS_LANG=language_territory.AL24UTF8SS
```

AL24UTF8SS is the setting for the UTF-8 character set.

Refer to Section 4.1.1, “The language environment variables” for more information on the language environment variables.

4.3.3 Font aliasing on Windows platforms

There may be situations where you create an application or a report with a specific font but find that a different font is being used when you run that application. You are most likely to encounter this problem when using an English font (such as MS Sans Serif or Arial) in a non-Western European environment. This occurs because Developer/2000 checks to see if the character set associated with the font matches the character set specified by the language environment variable. If the two do not match, Developer/2000 automatically substitutes the font with another font whose associated character set matches the character set specified by the language environment variable. This automatic substitution assures that the data being returned from the database gets displayed correctly in the application.

**Note:** If you enter local characters using an English font, Windows does an implicit association with another font.

There may be cases, however, where you do not want this substitution to take place. You can avoid this substitution by mapping all desired fonts to the WE8ISO8859P1 character set in the font alias file. For example, if you are unable to use the Arial font in your application, add the following line to your font alias file:

```
Arial......=Arial...WE8ISO8859P1
```

Refer to Section 4.1.1, “The language environment variables” for more information on the language environment variables.
4.4 Sorting Character Data

When you are designing multilingual applications, you want to sort character data according to the alphabetic conventions of a particular language rather than according to the characters’ binary values. The SQL function NLSSORT makes it possible to do this.

4.4.1 Comparing strings in a WHERE clause

Strings in a WHERE clause are compared according to the characters’ binary values: one character is considered greater than another if it has a higher binary value in the database character set. However, because the sequence of characters based on their binary values does not match the alphabetic sequence for a particular language, these comparisons yield incorrect results.

For example, suppose you have a column called COL1 that contains the values ABC, ABZ, BCD, and ÄBC. The database character set is ISO 8859/1. You write the following query:

```
SELECT COL1 FROM TAB1 WHERE COL1 > 'B'
```

The query returns BCD and ÄBC.

Now suppose you write this query:

```
SELECT COL1 FROM TAB1 WHERE NLSSORT(COL1) > NLSSORT('B')
```

If the language component of the language environment variable is set to German, the query returns BCD. If the language component of the language environment variable is set to Swedish, the query returns BCD and ÄBC.

4.4.2 Controlling an ORDER BY clause

If the language component of the language environment variable is set correctly, it is not necessary to use NLSSORT in an ORDER BY clause.

The following query yields a correct result:

```
SELECT ENAME FROM EMP
ORDER BY ENAME
```

4.5 Screen Design Considerations

When you are designing multilingual applications, remember to leave extra space in the base screen design for widgets and boilerplate labels. To accommodate multiple character sets and allow for expansion caused by translation, a rule-of-thumb is to leave 30% white space around fields, borders, and boilerplate text.
Specifically:
- Prompt on left of field: Allow for 30% expansion to the left of the prompt.
- Prompt above field: Allow for 30% expansion to the right of the prompt.
- Buttons, checkboxes, radio groups, and poplists: Allow for 30% expansion.
- Form titles: Size any bounding box so the title can expand to the right by 30%.
- Display-only fields: Size 30% wider than needed for base language.
- All widgets: Make widgets large enough to accommodate translation. For example, buttons should be large enough to hold translated labels. Check button height as well as length to be sure the height of the button will accommodate the tallest character you need to use. Calculate pixels needed to render Kanji characters.

### 4.6 Bidirectional Support

*Bidirectional support* enables you to design applications in Middle Eastern and North African languages whose natural writing direction is right-to-left. Bidirectional support enables you to control:

- layout direction, which includes displaying items with labels to the right of the item and correct placement of checkboxes and radio buttons.
- reading order, which includes right-to-left or left-to-right text direction.
- alignment, which includes switching point-of-origin from upper left to upper right.
- initial keyboard state, which controls whether local or Roman characters will be displayed when the user enters data. (The user can override this setting.)

When you are designing bidirectional applications, you may wish to use the language environment variables `DEVELOPER_NLS_LANG` and `USER_NLS_LANG` rather than `NLS_LANG`. Refer to Section 4.1.1, “The language environment variables” for more information on the language environment variables.

#### 4.6.1 Bidirectional support in Form Builder

Four properties are used to specify the appearance of objects in bidirectional applications: direction, alignment, reading order, and initial keyboard state.

Direction is an umbrella property that provides as much functionality for each object as possible. For all objects except text items and display items, the Direction property is the only bidirectional property, and its setting controls the other aspects.
of bidirectional function. (List items, however, include an Initial Keyboard State property.)

Text items and display items do not have a Direction property; instead, you can specifically set Alignment, Reading Order, and Initial Keyboard State properties for these items.

When the bidirectional properties are set to Default, those properties inherit their values from the natural writing direction specified by the language environment variable. In most cases, this will provide the desired functionality. You only need to specify the bidirectional properties when you want to override the inherited default values.

Inheritance for bidirectional properties is as follows:

<table>
<thead>
<tr>
<th>Default setting derives value from this object</th>
</tr>
</thead>
<tbody>
<tr>
<td>Form</td>
</tr>
<tr>
<td>All objects, such as Alert, Block, LOV, Window, and Canvas-view</td>
</tr>
<tr>
<td>All items, such as Text Item, Display Item, Checkbox, Button, Radio Group, and List Item</td>
</tr>
</tbody>
</table>

Most properties related to bidirectional function can be retrieved and set programmatically. For more information, see the appropriate built-in subprogram description. For example, for information about getting the value of the Direction property for buttons, refer to the description for GET_ITEMPROPERTY.

### 4.6.2 Bidirectional support in Report Builder

Four properties are used to specify the appearance of objects in bidirectional applications: Align, Direction (for an object), and Direction (for the report). The bidirectional properties are added to objects in the following hierarchy:

Module

Boilerplate

Field

External Boilerplate
4.8 Translating Your Applications

---

4.7 Translating the Designer’s User Interface

If you wish to use the Form Builder, Report Builder, or Graphics Builder Designers in a language other than English, simply specify the correct language and territory in the language environment variable. Messages, menus and menu items, dialog boxes, prompts and hints, and alerts are displayed in the appropriate language and numbers and dates in default values, ranges, and parameters are displayed in the appropriate format. If the appropriate message file is not available, the default is the US message file.

Refer to Section 4.1.1, “The language environment variables” for more information on the language environment variables.

4.8 Translating Your Applications

In any Developer/2000 application, the user sees the following:

- error messages from the database.
- runtime error messages produced by Form Builder, Report Builder, and/or Graphics Builder.
- messages and boilerplate text defined as part of the application.

If the language environment variable is set correctly and the appropriate message files are available, translation of messages in the first two categories is done for you. To translate messages in the third category, use one of the methods described in the following sections.

Refer to Section 4.1.1, “The language environment variables” for more information on the language environment variables.

4.8.1 Translating your applications using Translation Builder

Translation Builder can help you translate form menus, boilerplate text, item labels, messages, and hints defined on item property sheets in your Developer/2000 applications. Using Translation Builder, you can generate separate executable files for each language.
If you plan to use Translation Builder to translate your application, develop the application in the following stages:

- Create one basic form definition (.FMB, .MMB) in the source language.
- Use Translation Builder to extract strings for translation, translate the strings into one or more languages, and store the translated strings back into the form definition.
- Manually translate messages in PL/SQL libraries. (Refer to Section 4.8.2, “Using PL/SQL libraries for strings in code” for more information on translating messages that are displayed programmatically.)
- Use the Form Builder Generate component to generate an executable version of the form (.FMX) for each target language.

At runtime, the form is displayed in the language whose .FMX file is first in your search path.

For example, if you create an application in four languages and then change a button label, you would do the following:

- Make the change in Form Builder Designer and save it in the .FMB file.
- Use Translation Builder to translate the new button label and insert the new messages into the .FMB file.
- Regenerate to create an .FMX file containing the new button label.

### 4.8.2 Using PL/SQL libraries for strings in code

While Translation Builder helps you translate strings in your application’s user interface, it cannot pull out string constants in PL/SQL triggers and procedures. Manual translation is required for constant text within a PL/SQL block because that text is not clearly delimited, but is often built up from variables and pieces of strings. To translate these strings, you can use PL/SQL libraries to implement a flexible message structure.

Refer to Section 4.8.1, “Translating your applications using Translation Builder” for more information on translating strings in your application’s user interface.

You can use the attachable PL/SQL libraries to implement a flexible message function for messages that are displayed programmatically by the built-in routines MESSAGE or CHANGE_ALERT_MESSAGE, or by assigning a message to a display item from a trigger or procedure. The library can be stored on the host and dynamically attached at runtime. At runtime, based on a search path, you can pull in the
library attached to the form. For example, a library might hold only the Italian messages:

```sql
FUNCTION nls_appl_mesg(index_no NUMBER)
RETURN CHAR
IS
    msg CHAR(80);
BEGIN
    IF index_no = 1001 THEN
        msg := 'L’impiegato che Voi cercate non esiste...';
    ELSIF index_no = 1002 THEN
        msg := 'Lo stipendio non puo essere minore di zero.';
    ELSIF ...
    ELSE
        msg := 'ERRORE: Indice messaggio inesistente.';
    END IF;
    RETURN msg;
END;
```

A routine like this could be used anywhere a character expression would normally be valid. For example, to display an alert with the appropriately translated application message, you might include the following code in your form:

```sql
Change_Alert_Message('My_Error_Alert', nls_appl_mesg(1001));
n := Show_Alert('My_Error_Alert');
```

To change the application to another language, simply replace the PL/SQL library containing the `nls_appl_mesg` function with a library of the same name containing the `nls_appl_mesg` function with translated text.
With Developer/2000, deploying a single application on multiple platforms—Windows, Motif, Macintosh, the Web, even character-mode—is simply a matter of re-compiling. Developer/2000 automatically translates the standard control objects (buttons, check boxes, radio buttons, and so on) to the appropriate format for your target platforms. With careful pre-development planning, you can create a single application that satisfies users across environments, providing each with the native look and feel they expect.

This chapter helps you anticipate the issues you will face when developing portable applications and provides suggestions for ensuring that you can move your application across platforms with ease.

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Section 5.1, “Before You Begin”</td>
<td>Presents some high-level questions you should answer before developing a portable application.</td>
</tr>
<tr>
<td>Section 5.2, “Designing Portable Forms”</td>
<td>Addresses issues of portability with respect to the GUI and the operating system. Also discusses an approach for cross-platform development, as well as considerations unique to the character-mode environment.</td>
</tr>
<tr>
<td>Section 5.3, “Designing Portable Reports”</td>
<td>Discusses developing a report for maximum portability.</td>
</tr>
<tr>
<td>Section 5.4, “Designing Portable Displays”</td>
<td>Discusses developing a display for maximum portability.</td>
</tr>
</tbody>
</table>
For specific information on porting applications to the Web, see Chapter 2, “Deploying Applications on the Web”. If you’re using Project Builder to manage your application, see “Managing projects and project documents across multiple platforms”.

5.1 Before You Begin

Before you begin designing any application—not just those you intend to port—it’s important that you take time to think about the needs you are trying to address with your application. At a minimum, you should have answers to the following questions:

- Which platforms are you supporting? If you plan to deploy on more than one platform, you must consider issues such as fonts, colors, layout, screen size, and screen resolution, to name a few. Section 5.2.1, “Considering the GUI” helps you tackle these issues.

- Is character-mode support required? If so, your options are considerably limited. Consult Section 5.2.4, “Designing forms for character-mode” or Section 5.3.1, “Designing a report for character-mode environments” for some recommendations.

- What displays must you accommodate? Displays can vary a great deal, even on the same deployment platform. See Section 5.2.1.2, “Considering monitors” for a discussion on the limitations monitors can impose.

- Will your application rely on user exits or foreign functions? You’ll probably have to rewrite them for each of your target platforms. Section 5.2.2.1, “Including user exits” offers some suggestions and workarounds.

5.2 Designing Portable Forms

Whether you’re designing a new form for multiple platforms or preparing an existing form for a new environment, the issues you face fall into the same two key areas, described in the following sections:

- Section 5.2.1, “Considering the GUI”

- Section 5.2.2, “Considering the operating system”

If you’ve never developed for multiple platforms before, you may also wish to read Section 5.2.3, “Strategies for developing cross-platform forms” for some recommendations on how to approach cross-platform development. If you’re developing for character-mode, see Section 5.2.4, “Designing forms for character-mode” for considerations unique to that environment.
5.2 Designing Portable Forms

5.2.1 Considering the GUI

When developing a portable application, the first thing you must decide is whether the GUI should look the same across all platforms, or if your users expect the application to inherit the native look-and-feel of their own environment. In most cases, you’ll probably opt for the latter approach. However, if users are likely to use the application on multiple platforms, they’ll probably want it to look the same on all of them, ignoring local conventions. The only way to determine this is to interview your users, paying close attention to how they work and which tasks they’re trying to perform. (See Section 3.1.2, “Defining user requirements” for suggestions on how to determine users’ needs.)

Once you’ve made your decision, the next step is to create an object library for each platform you’re supporting. An object library is a set of objects and standards that you create; each object or standard can determine the appearance and layout of an entire frame, window, or region. When housed in an object library, these objects become available to all the developers on your project or site, thus ensuring that even developers working at different locations can produce an application—or different modules within the same application—with a common look and feel.

To fully exploit the power of the object library, it’s a good idea to create one library for each of your target platforms. To help you populate your libraries, Form Builder provides the Oracle Applications Object Library, a set of layouts and items that function well in all of Developer/2000’s GUI deployment environments (Windows 95, Macintosh, Motif), as well as character-mode. Test these items and objects one by one on each of your platforms. You should be able to add most of the objects to your libraries without modification, although some may need slight adjustment to meet platform-specific requirements.

Section 5.2.3, “Strategies for developing cross-platform forms”, provides more details on how to incorporate your object libraries into an overall development strategy.

5.2.1.1 Choosing a coordinate system

For GUI terminals, use the Real Inch, Real Centimeter, or Real Point coordinate systems. These systems allows you to size your objects to the exact shape you want instead of being snapped to the nearest character cell size.

If you’re designing for character-mode, use the Character coordinate system and turn on the grid snap. This will ensure that your objects’ sizes are in multiples of the character cell size. See Section 5.2.4, “Designing forms for character-mode” for more information on designing character-mode applications.
5.2.1.2 Considering monitors

Even on the same platform, monitors of different sizes and resolutions can greatly impact the usability of your application. For example, while a 6 pt. font on a laptop running Windows 95 is unreadable, the same font on a 17-inch monitor is perfectly acceptable. The only way to be certain your application is truly portable is to thoroughly test your application on each of the monitors in the deployment environment.

If there are several different sized monitors in your deployment environment, design for the smallest size. Taking the time to find out which monitors your users have—and how many use each size—can help you plan your application more effectively. For example, if your mobile sales force uses laptops for lead tracking and sales management applications, but everyone else uses 17-inch SVGA terminals, you can simplify your task by restricting the window size of only the two critical laptop applications.

<table>
<thead>
<tr>
<th>Windows</th>
<th>Motif</th>
<th>Macintosh</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size is determined by screen resolution, not by absolute measurement. For example, widgets developed on a 96 dots per inch (dpi) 17-inch monitor appear smaller than the same widgets displayed on a 20-inch 96 dpi monitor, even though the measurement systems appear to be the same. In other words, an inch is not always an inch on Windows.</td>
<td>Many Motif users are limited to gray-scale monitors, so you can’t rely on color for those users.</td>
<td>Screen resolution is always 72 dpi. To display more objects on the screen, the user must have a bigger monitor.</td>
</tr>
</tbody>
</table>

Table 5–1 Platform restrictions: Monitors

5.2.1.3 Using color

Restrain your use of color to three or four basic colors that work well together. Colors that are typically available on many platforms include blue, red, magenta, cyan, green and yellow.

Using too many colors can exceed the system’s maximum color limit and cause background objects to snap to strange colors, leaving only the foreground color
intact. Be sure to test your color combination on all target systems, including monochrome, gray-scale monitors, to make sure they work as expected.

<table>
<thead>
<tr>
<th>Windows</th>
<th>Motif</th>
<th>Macintosh</th>
</tr>
</thead>
<tbody>
<tr>
<td>Widgets can be one of 16 colors defined in the system color palette. If you assign another color, the widget snaps to the closest of the sixteen.</td>
<td>Many Motif users are limited to gray-scale monitors; do not use color to make important distinctions.</td>
<td>Many Macintosh users are limited to gray-scale monitors; do not use color to make important distinctions.</td>
</tr>
</tbody>
</table>

**Table 5–2  Platform restrictions: Color**

### 5.2.1.4 Resolving font issues

Fonts play a fundamental role in the user’s sense of familiarity and comfort with a GUI system. Table 5–3 lists the recommended font for each GUI platform:

<table>
<thead>
<tr>
<th>Windows</th>
<th>Motif</th>
<th>Macintosh</th>
</tr>
</thead>
<tbody>
<tr>
<td>MS Sans Serif</td>
<td>Helvetica</td>
<td>Geneva or Chicago</td>
</tr>
</tbody>
</table>

**Table 5–3  Platform recommendations: Fonts**

When developing a portable application, decide early how you’ll use font styles such as boldface, italics, and underlining. (In general, you shouldn’t need either underlining or italics; use boldface sparingly, and only for emphasis.) You should also standardize the type size of different display objects. For example, making all labels 10 points will help if you need to translate a font on a different platform.

To meet users’ expectations, a ported application must be rendered in the expected font on each platform. To achieve this, you must translate the fonts between platforms using either of these methods:

- Defining aliases for fonts on each deployment platform
- Defining port-specific classes

The next two sections briefly outline these processes.

**Note:** On Motif, each different size of a given font is considered a separate entity that must be explicitly installed from the font file. For example, suppose you want to port a Windows-based form containing 10, 12, and 28 point Arial fonts to Motif. Rather than simply verifying that Arial has been installed on Motif, you must ensure that each of the desired point sizes—10, 12, and 28—have been installed as
well. If Developer/2000 can’t find the font it needs on the target platform, it substi-
tutes another font using a platform-specific “closest match” algorithm.

5.2.1.4.1 Defining font aliases

Developer/2000 provides a font alias file for each platform (UIFONT.ALI, in the
ORACLEHOME/BIN directory). In most cases, the file ensures that fonts appear con-
sistently across platforms. However, if you employ custom or non-standard fonts in
your applications, some of them may not be recognized on all target platforms. You
can tailor the Developer/2000 font alias file to define substitutions for the fonts that
are not recognized.

Enter each line in the file in this format:

source_font = destination_font

For each font, you can specify these attributes:

<face>.<size>.<style>.<weight>.<width>.<character_set>

Example 1:

When porting from MS Windows to Macintosh, change all MS Sans Serif 10-point
font to Geneva 12-point font:

“MS Sans Serif”.10=Geneva.12

Example 2:

When porting from MS Windows to Motif, change all MS Sans Serif fonts to Helvet-
ica:

“MS Sans Serif”=Helvetica

See the Form Builder or Report Builder online help for more information and exam-
pies of font mapping.

5.2.1.4.2 Using classes

When you require greater control over your font aliasing, use classes. For example,
suppose you want your poplists and text items to have different fonts on Motif,
rather than just imposing a strict conversion of MS Sans Serif to Helvetica. To
achieve this:

1. Create two classes, one for poplists and the other for text items.

2. On MS Windows, specify that both classes use MS Sans Serif as the font in Win-
dow.olb. (See Section 5.2.3.1, “Creating a single source” for information on
Window.olb.)
3. In Motif.olb, specify that the poplist class uses the Helvetica 9-point font; specify that the text item class uses Helvetica 11-point.

This approach allows you to customize the font used for each class of objects in your application, thus providing a higher level of flexibility.

5.2.1.5 Using icons

Icons are platform-specific. If you use iconic buttons in your application, create a separate icon directory for each platform. Use the same names for the icons on each platform and set the respective environment variable to point to the icon directory. On MS Windows and Motif, this variable is `TK25_ICON`. (On Macintosh, icons are compiled in the modules themselves.)

If you include icons in your application, keep the following in mind:

- Icons rendered on small monitors (like laptops) can be too small to read.
- Certain icons have special meanings on certain platforms. For example, the trash can on the Macintosh is reserved for the delete function and cannot be used for some other purpose.

5.2.1.6 Using buttons

In MS Windows, a button’s moat (the emphatic border around a button to designate a default) is very small compared to that on Motif. Therefore, buttons appear to shrink when run on Motif and Macintosh. On Motif, you can avoid this by modifying the Motif resource file, `Tk2Motif` in `ORACLE_HOME/BIN`. (Oracle uses Motif resource files to control the visual appearance of UNIX-based applications.)

1. Locate the `Tk2Motif` file for your display type:
   - `gs` (gray scale)
   - `bw` (black and white)
   - `rgb` (color)

2. Edit the `Tk2Motif` file and set the `Tk2Motif expandNonDefaultButtons` property to `True`.

On Macintosh, consider displaying an alternative set of buttons for these environments. In general, always provide enough space in your Windows buttons to accommodate the larger button size in Motif and Macintosh.

To maximize portability, make all buttons non-navigable. In Windows and Motif, clicking a button means the user actually navigates to the button; this is not the case in Macintosh. Because triggers are often dependent upon button navigation,
this difference across platforms can create significant behavioral differences in your application.

**Note:** Making Windows and Motif buttons non-navigable is an excellent example of the kind of trade-off you might have to make if consistency across platforms is more important than adhering to standard platform behavior.

### 5.2.1.7 Creating menus

The placement and behavior of menus varies across platforms, as shown in Table 5–4:

<table>
<thead>
<tr>
<th>Windows</th>
<th>Motif</th>
<th>Macintosh</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supports Multiple Document Interface (MDI) and Single Document Interface (SDI). In MDI, all windows belonging to an application are contained in a single window, and there is only one menu for the entire application. SDI is similar to Motif in that each window has its own menu.</td>
<td>Every window has a menu attached. The menu on a parent window may or may not be repeated on child windows.</td>
<td>Single menu across the top of the screen.</td>
</tr>
</tbody>
</table>

*Table 5–4 Platform restrictions: Menus*

If you are using a version of Windows that supports MDI and you want your applications to look the same across all platforms, specify in Motif that you do **not** want to repeat the parent window menu on child windows. Then you can design the parent window menu to look exactly like that on MS Windows and/or Macintosh.

**Note:** To prevent the screen from flashing when switching between form module windows, combine all the menu options into one single menu application and use the `SET_MENU_ITEM_PROPERTY` built-in to dynamically enable/disable the respective menu items accordingly.
5.2.1.8 Creating the console

Like menus, the placement and behavior of the console also varies across platforms, as shown in Table 5–5:

<table>
<thead>
<tr>
<th>Windows</th>
<th>Motif</th>
<th>Macintosh</th>
</tr>
</thead>
<tbody>
<tr>
<td>Appears at the bottom of the MDI window only.</td>
<td>Appears on the user-specified window.</td>
<td></td>
</tr>
</tbody>
</table>

Table 5–5 Platform restrictions: Console

To achieve consistency across platforms, place the console on the parent window in your Motif application to emulate the behavior of Macintosh and MDI Windows applications.

5.2.1.9 Miscellaneous

- When building a form for multiple platforms, right-align all prompts. Text often expands when ported to other platforms, and left-aligned prompts can cause fields to shift, creating a ragged margin.

- To provide complex functionality that is completely portable across platforms, employ one or more Developer/2000 reusable components in your application. These reusable components are provided in the Developer/2000 Demos and Add-ons, to help you build applications upon such powerful features as:
  - Navigator (Explorer) style interface
  - Wizard style interface that mimics the Wizards in Developer/2000 and other Windows 95 products
  - Calendar window that automatically displays the calendar according to the NLS settings currently in effect
  - Image and icon files
  - Standard menu in the Windows style

Refer to “Reusable Components” (under Developer/2000 Demos) in the Form Builder online help for more information.
5.2 Designing Portable Forms

Table 5–6, “Platform restrictions: General” lists other miscellaneous issues related to porting GUIs:

<table>
<thead>
<tr>
<th>Windows</th>
<th>Motif</th>
<th>Macintosh</th>
</tr>
</thead>
<tbody>
<tr>
<td>A known positioning problem causes two lines forming a right angle on VGA screens to actually overlap on SVGA. Use bevels to avoid this problem.</td>
<td></td>
<td>Users cannot use the keyboard to navigate to anything except text items.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bevels are not available.</td>
</tr>
</tbody>
</table>

5.2.2 Considering the operating system

No application is truly portable if it depends on functionality unique to a particular operating system. Here are some general rules to keep in mind:

- Avoid port-specific terminology when writing messages. For example, a message like “Press F1 for help” is not portable.

- Do not hardcode path names; path names vary across platforms. Instead, use environment variables to enable Form Builder to find your files during runtime.

Suppose you need to read an image file called OPEN.BMP from your form. In a Windows-only application, you could simply code the path name in the call to READ_IMAGE_FILE:

```java
Read_Image_File('c:\orawin95\myapp\open.bmp', 'BMP', 'block1.image3');
```

If you want the application to be portable, however, hardcoding won’t work, since the name of the path is different on each platform. Instead, you can use an external variable to represent the path name.

For example, in Windows95 or WindowsNT:

1. Create a registry entry called path_var under the ORACLE key; in UNIX, create a shell variable also named path_var.

2. Use the GETVAR procedure in the TOOL_ENV package to retrieve the value path_var using this platform-independent method:

```java
path_var varchar2(255);
...
Tool_env.getvar('MYPATH', path_var);
Read_Image_File(path_var||'open.bmp', 'BMP', 'block1.image3')
```

Table 5–6 Platform restrictions: General
The platform-specific path name, represented by the variable `path_var`, is appended to the name of the image file, `OPEN.BMP`. On Windows 95, `path_var` resolves to the path name `C:\ORAWIN95\MYAPP\`. On UNIX, `path_var` is something like `/oracle_home/myapp/`.

- Anything called through the HOST built-in procedure. Host commands execute port-specific operating system commands. To make your application easier to port:
  1. Create a separate procedure library (`.PLL`) for each platform.
  2. Place all operating system commands in the appropriate procedure library.
  3. Create a generic procedure library.
  4. Rewrite the script file for each platform, ensuring that each script has the same name.
  5. In your form module, make sure all calls refer to the generic procedure library.
  6. Before compiling on a given platform, copy that platform’s `.PLL` to the generic procedure library.
  7. Compile.

Section 5.2.3, “Strategies for developing cross-platform forms” explains how this handling of procedure libraries fits into the recommended development strategy for portable applications.

- Context-sensitive help is not portable. If your application uses native context-sensitive help, replace it with Developer/2000’s portable help component, the Online Help Class. This component enables you to provide context-sensitive help in your application similar to Windows 95 help.

The component is built using Form Builder and PL/SQL native capabilities, so it is portable to all Forms-supported platforms. Because the help text you create is stored in the database, it is accessible to all users, and updates are immediately available to everyone.

To use the Online Help Class in your application:
  1. Install the database objects.
  2. Create the help text for your application.
  3. Attach a PL/SQL library and add code in your key-help trigger to call the help as required.
Refer to the help topic “About the Online Help Class” under Developer/2000 Demos in the Form Builder online help for step-by-step instructions.

- Avoid including the platform-specific methods listed in Table 5–7.

<table>
<thead>
<tr>
<th>Windows</th>
<th>Motif</th>
<th>Macintosh</th>
</tr>
</thead>
<tbody>
<tr>
<td>VBX controls</td>
<td>Calls to loadable libraries (ORA_FFI)</td>
<td>AppleEvents</td>
</tr>
<tr>
<td>OLE containers</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ActiveX (OCX)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DLLs (ORA_FFI)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Table 5–7  Platform specific methods*

These objects leave placeholders (empty spaces) on the platforms that do not support them. If you must include these objects in your application, see Section 5.2.3.4, “Hiding objects” for information on how to prevent the placeholders from appearing.

### 5.2.2.1 Including user exits

A *user exit* is a 3GL program you write yourself and then link into a form at compile time. User exits are always port-specific.

Before calling a 3GL program from your portable form, verify that the information and processes on which the program relies are available on all platforms. For example, a program that depends on information from the Windows registry can’t access this information on other platforms, which means you may have to re-design the program or abandon it entirely.

Rather than accessing a 3GL program through the user interface exit, consider the use of the ORA_FFI built-in package (Oracle Foreign Function Interface). If you use the user exit interface to access your foreign functions, you must re-link the user exits or replace the DLL for each platform each time a 3GL program changes. Because ORA_FFI allows you to call foreign functions through a PL/SQL interface using PL/SQL language conventions, re-linking isn’t required when you modify a program. For this reason, ORA_FFI is the preferred method for accessing 3GL programs from your forms.

### 5.2.3 Strategies for developing cross-platform forms

This section introduces some techniques you can use to develop portable forms:
Section 5.2.3.1, “Creating a single source” describes an architecture for creating a single source that delivers maximum functionality on each of your deployment platforms.

Section 5.2.3.2, “Subclassing visual attributes” discusses the importance of explicitly subclassing the visual attributes stored in your object libraries.

Section 5.2.3.3, “Using the get_application_property built-in” discusses the use of this Form Builder built-in when developing portable applications.

Section 5.2.3.4, “Hiding objects” provides sample code for removing the placeholders that appear when an object is not valid on a particular platform.

### 5.2.3.1 Creating a single source

While it may be tempting to consider creating a single source that aims at the lowest common denominator for all deployment platforms, this strategy severely limits the aesthetics you can provide in your application. A more effective strategy is to create a single source that delivers applications in each platform’s native look-and-feel. The architecture depicted in illustrates how you might accomplish this:

![Diagram showing port-specific implementation](image_url)

**Figure 5-1  Port-specific implementation**

To model your application on this architecture:

1. Create an object library for all standards and objects (**Porting.olb**). Refer to Section 3.2.2.1, “Using object libraries” for information on using object libraries.

2. Create separate object libraries for each deployment platform (**Window.olb**, **Motif.olb**, **Mac.olb**, **Charmode.olb**, **Window.pll**, **Motif.pll**, **Mac.pll**).

3. Create a common library for port-specific code (**Porting.pll**).
4. Create separate libraries for each platform’s port-specific code (Window.pll, Motif.pll, Mac.pll, Charmode.pll).

5. In each platform’s UI repository (.OLB) and library (.PLL), develop code to handle the application objects in the manner ideal for that particular platform. Use the same name for a given object in each UI repository and library.

6. Write your application, referring to the standards and objects in the repository and to the port-specific code in the library.

7. When you’re ready to compile your application for a particular platform, copy that UI’s repository and library to Porting.olb and Porting.pll and compile.

5.2.3.2 Subclassing visual attributes

Visual attributes are the font, color, and pattern properties you set for form and menu objects. By carefully defining the visual attributes of your form objects, you can ensure that users on each platform enjoy the native look-and-feel unique to that environment.

Many Form Builder objects, such as items and canvases, refer to visual attributes to define their appearance. Visual attributes must be defined in the same module as the object that refers to them.

Visual attributes are usually stored in an object library. It’s a good idea to create subclasses of these visual attributes in each module. When you subclass an object from an object library in your form, any changes made to the library object are automatically applied to the form object. However, this does not apply to changes made to the library object’s visual attributes. So, by subclassing, rather than coping the visual attributes, you ensure that your modules always reflect the latest definition of the visual attributes.

5.2.3.3 Using the get_application_property built-in

The GET_APPLICATION_PROPERTY built-in function returns information about your application, allowing you to react dynamically at runtime based on the settings of one or more of these variables:

- DISPLAY_HEIGHT and DISPLAY_WIDTH. Specifies how big the current display is. The unit depends on how you have set up the form coordinate system.

- OPERATING_SYSTEM. Specifies the name of the platform on which the application currently is running (MSWINDOWS, WIN32COMMON, MACINTOSH, SunOS, VMS, UNIX, or HP-UX).
5.2 Designing Portable Forms

- USER_INTERFACE: Specifies the name of the user interface technology on which the application is currently running (WEB, MOTIF, MACINTOSH, MSWINDOWS, MSWINDOWS32, PM, X, VARCHAR2MODE, BLOCKMODE, or UNKNOWN).

Depending on the value of a variable, you can dynamically hide objects that are not available on that deployment platform, reposition other objects to take up that space and, if necessary, alter the attributes of an object to suit the standards on that deployment platform. See Section 5.2.3.4, “Hiding objects” for more information.

5.2.3.4 Hiding objects

To prevent users from seeing placeholders on platforms that do not support OLE, VBX, and ActiveX objects, you can put these objects in a separate window invoked from the menu or a button and dynamically enable/disable the menu item or button. Or you can use this code fragment to hide/show the port-specific objects and reposition other objects to take their place:

```sql
WHEN-NEW-FORM-INSTANCE trigger:
  declare
  ui  varchar2(15) ;
  begin
  ui := get_application_property (user_interface);
  if ui = 'CHARMODE' or ui = 'MOTIF' then
    set_item_property ('VBXObject1', displayed, property_false);
    set_item_property ('OLEOBJECT1', displayed, property_false);
    set_item_property ('TEXTITEM1', position, 43, 4);
  end if;
  end;
```

Note: Item prompts are automatically hidden when you hide the associated item.

5.2.4 Designing forms for character-mode

If you are creating an application for both character-mode and bit-mapped environments, single-sourcing is probably not the best approach. Developing for the lowest common denominator, character-mode, deprives your GUI users of the ease of use
associated with bit-mapped controls. The “Bit-map Only” column in Table 5–8 lists the functions you’d have to avoid:

<table>
<thead>
<tr>
<th>Character Mode</th>
<th>Bit-map Only</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boxes</td>
<td>Images</td>
</tr>
<tr>
<td>Horizontal lines</td>
<td>Color</td>
</tr>
<tr>
<td>Vertical lines</td>
<td>Drawings</td>
</tr>
<tr>
<td>ASCII text</td>
<td>Ellipses</td>
</tr>
<tr>
<td>Boldface text</td>
<td>Drill-down buttons (reports)</td>
</tr>
<tr>
<td>Underlines</td>
<td>Italicized text</td>
</tr>
<tr>
<td></td>
<td>Bit-map patterns</td>
</tr>
<tr>
<td></td>
<td>Diagonal lines</td>
</tr>
<tr>
<td></td>
<td>Multimedia support</td>
</tr>
<tr>
<td>Images</td>
<td>Images</td>
</tr>
<tr>
<td>Color</td>
<td>Color</td>
</tr>
<tr>
<td>Drawings</td>
<td>Drawings</td>
</tr>
<tr>
<td>Ellipses</td>
<td>Ellipses</td>
</tr>
<tr>
<td>Drill-down buttons</td>
<td>Drill-down buttons (reports)</td>
</tr>
<tr>
<td>Italicized text</td>
<td>Italicized text</td>
</tr>
<tr>
<td>Bit-map patterns</td>
<td>Bit-map patterns</td>
</tr>
<tr>
<td>Diagonal lines</td>
<td>Diagonal lines</td>
</tr>
<tr>
<td>Multimedia support</td>
<td>Multimedia support</td>
</tr>
</tbody>
</table>

Table 5–8 Character-mode vs. bit-mapped environments

While there are methods for disabling these GUI functions in a character-mode environment, this task can be extremely time-consuming and frustrating for you. So, if you know from the beginning that you have to support these two widely disparate sets of users, it’s best for everyone—you and your users—to simply create two entirely separate applications.

It’s much easier to develop for character-mode if you make Form Builder look like character-mode as much as possible. Table 5-9 lists some recommended property settings:

<table>
<thead>
<tr>
<th>Property</th>
<th>Recommendations/Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boilerplate font</td>
<td>Windows: FixedSys, Regular, 9 pt</td>
</tr>
<tr>
<td></td>
<td>Motif: Font=Fixed, Size=12.0, Weight=Medium, Style=Plain</td>
</tr>
<tr>
<td>Coordinate information</td>
<td>Coordinate system: Real^1</td>
</tr>
<tr>
<td></td>
<td>Real Unit: Point</td>
</tr>
<tr>
<td></td>
<td>Character Cell Width: 6</td>
</tr>
<tr>
<td></td>
<td>Character Cell Height: 14</td>
</tr>
</tbody>
</table>

Table 5-9 Property settings that resemble character-mode
5.2 Designing Portable Forms

As you develop your application strictly for character-mode, keep the following in mind:

<table>
<thead>
<tr>
<th>Property</th>
<th>Recommendations/Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>View</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Grid: on</td>
</tr>
<tr>
<td></td>
<td>Grid Snap: on</td>
</tr>
<tr>
<td></td>
<td>Show Canvas: off</td>
</tr>
<tr>
<td>View→Settings→Ruler</td>
<td>Units: Character cells</td>
</tr>
<tr>
<td></td>
<td>Character Cell Size Horizontal: 6</td>
</tr>
<tr>
<td></td>
<td>Character Cell Size Vertical: 14</td>
</tr>
<tr>
<td></td>
<td>Grid Spacing: 1</td>
</tr>
<tr>
<td></td>
<td>Snap Points per Grid Spacing: 2</td>
</tr>
</tbody>
</table>

Table 5-9 Property settings that resemble character-mode

1 Improves portability of the form from character-mode to bit-mapped environments. If the form will be deployed in character-mode only, use the Character coordinate system.

As you develop your application strictly for character-mode, keep the following in mind:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Recommendations/Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>General</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Remember that everything is in monospace font.</td>
</tr>
<tr>
<td></td>
<td>Create keyboard equivalents for each widget, even when the widget does not have the current focus.</td>
</tr>
<tr>
<td></td>
<td>Avoid scrolling, as it is very hard to use.</td>
</tr>
<tr>
<td></td>
<td>Hide OLE, VBX, and ActiveX objects if you do not want users to see their placeholders.</td>
</tr>
<tr>
<td></td>
<td>Be sure that widgets have sufficient space to display themselves entirely, as all UI widgets are rendered in their character-mode equivalents.</td>
</tr>
<tr>
<td></td>
<td>Because users cannot move an LOV with a mouse, use the <code>set_lov_property</code> built-in to dynamically position the LOV.</td>
</tr>
<tr>
<td>Navigation</td>
<td>Since the user does not have a mouse, users cannot navigate between windows or forms from within the application. Provide buttons or menu options for navigating between forms.</td>
</tr>
<tr>
<td></td>
<td>Since windows cannot be repositioned with a mouse, ensure that a displayed window does not obscure the context required for that window. When the user is done with a window, disable the window programmatically, or set the window’s Remove On Exit property to true.</td>
</tr>
</tbody>
</table>
5.3 Designing Portable Reports

When preparing a report to run on multiple platforms, consider the following:

- **Fonts.** Not all font types, styles, and sizes are available on all target GUIs. You can handle this in one of two ways:
  - Use a font that you know exists on the target GUI or one that maps well to the default font of the target GUI.
5.3 Designing Portable Reports

- Modify the font mapping file, `UIFONT.ALI`, to ensure that the fonts map correctly. See Section 5.2.1.4.1, “Defining font aliases” for more details on using the `UIFONT.ALI` file.

  **Note:** Because screen font and printer font metrics are not always the same, your printed report may not look the same as it did on the screen. In particular, text fields can expand on the printed page, causing adjacent fields to shift and possibly creating new and unpredictable page breaks. To avoid this, use expand-only fields and be sure each field is large enough to accommodate the largest font reasonably possible.

- **Colors.** If possible, use a color that you know exists on the target GUI; otherwise, use one that maps well to the default color of the target GUI. The following colors are typically available on many platforms: blue, magenta, red, cyan, green, yellow. See Section 5.2.1.3, “Using color” for some recommendations on including color in portable reports.

- **DPI.** The dots-per-inch (DPI) that your monitor uses may not be the same as the DPI used by the person who runs the report. The DPI only affects how alpha-numeric characters word-wrap on the screen. If you design a report that may be displayed in the Previewer view, try to use the same DPI as the people who will run it.

- **Buttons.** If you provide buttons in a report, users viewing the report through the Previewer can press the buttons to display a multimedia object (sound, video, image) or to perform an action through PL/SQL code, such as drilling down to another report. See Section 5.2.1.6, “Using buttons” for some guidelines on creating portable buttons. Note that if you run a report containing buttons in character-mode, the buttons are simply ignored; they do not create a placeholder.

### 5.3.1 Designing a report for character-mode environments

Character-mode reports are often needed in environments where users need to send their report output to bulletin boards, spreadsheets, dump files, or to character-only printers. Character-mode output also provides a number of advantages:

- **Portability.** Because they are strictly ASCII or EBCDIC files, character-mode reports can be printed or exported anywhere.

- **Protecting printer investment.** Character-mode reports require no special formatting—unlike complicated postscript output—thus protecting your investment in older printers.
5.4 Designing Portable Displays

- **Printer code support.** Developer/2000 provides support for printer escape codes, which enable users to exploit printer-specific features at runtime, such as special font sizes, highlighting, and more. Refer to the Report Builder online help for information on printer definition files and printer codes.

- **Performance.** Character-mode reports run much faster than an equivalent bit-mapped report. Bit-mapped reports typically require more formatting time and have larger (Postscript, PCL5) output files.

5.3.1.1 Design considerations

Reports built for bit-mapped environments cannot easily be adapted to character-mode. If you know you will need to run a report in a character-mode environment, it is best to build the report as a character-mode report. However, if you must convert a bit-mapped report to character mode, refer to “Converting a bit-mapped report to ASCII (character mode)” in the Report Builder online help.


5.4 Designing Portable Displays

If you have standalone graphics—graphics that are not part of a container application such as a form or report—porting is fairly straightforward. Most graphics, however, are embedded within forms and reports, which can introduce problems when moving across platforms. When developing graphics for multiple environments, observe the following guidelines:

- To ensure that text scales uniformly—especially when the graphic is embedded in a form or report—use a scalable truetype font and set the Scalable Fonts flag to true for all text objects except chart labels. The Scalable Fonts flag is not available for text labels in a chart. As soon as a chart is updated, fonts are re-set to their original size. Thus, choose a font and size for chart labels that is legible at the greatest range of chart sizes. A good bet is small, sharp fonts that display well at 8-10 point sizes. Anything larger may cause your chart to become unreadable when embedded in a small chart area of a form or report.

- Timers and drag-and-drop code are supported only in standalone Graphics applications. If you include these functions in a form or report, they are ignored.

- Limit your use of colors to the core 16, which are available in the Designer (upper-left corner of the palette), as well as through their mnemonic names.
(red, green, blue, yellow, magenta, cyan, black, white, gray, darkgray, darkyellow, darkcyan, darkmagenta, darkblue, darkgreen, and darkred).

- Set colors through the layout editor, rather than through PL/SQL. Colors chosen in the layout editor are automatically adjusted to the nearest available color. Colors set through code can result in an error if the color is not available on your system at the current resolution.

- Isolate platform-dependent code with calls to the application property `og_get_ap_platform`, and to the built-in subprograms `og_append_directory` and `og_append_file`. Refer to the Graphics Builder online help for more information.
5.4 Designing Portable Displays
Taking Advantage of Developer/2000’s Open Architecture

This chapter offers guidelines to help you take advantage of Developer/2000’s open and extensible development environment.

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Section 6.1, “Working with OLE Objects and ActiveX Controls”</td>
<td>Describes Developer/2000’s support for component technologies and provides steps and guidelines for creating applications that include OLE objects and ActiveX controls.</td>
</tr>
<tr>
<td>Section 6.2, “Using Foreign Functions to Customize Your Applications”</td>
<td>Describes how to customize and extend your applications with 3GL foreign functions.</td>
</tr>
<tr>
<td>Section 6.3, “Using the Open API”</td>
<td>Introduces the Open API and explains how to use the Open API to build and modify Form Builder applications.</td>
</tr>
<tr>
<td>Section 6.4, “Designing Applications to Run against ODBC Datasources”</td>
<td>Discusses Developer/2000’s ODBC support and provides detailed steps and guidelines that describe how to run applications against ODBC datasources.</td>
</tr>
</tbody>
</table>
6.1 Working with OLE Objects and ActiveX Controls

This section describes what OLE and ActiveX are, and how you can exploit this technology within Developer/2000. This section includes these topics:

- Section 6.1.1, “What are OLE objects?”
  - Section 6.1.1.1, “Guidelines for using OLE objects”
  - Section 6.1.1.9, “Creating OLE objects”
  - Section 6.1.1.10, “Manipulating OLE objects”
  - Section 6.1.1.11, “OLE examples”

- Section 6.1.2, “What are ActiveX controls?”
  - Section 6.1.2.1, “Guidelines for using ActiveX controls”
  - Section 6.1.2.2, “Manipulating ActiveX controls”
  - Section 6.1.2.7, “Creating ActiveX controls”
  - Section 6.1.2.8, “ActiveX examples”

Note: Support for OLE and ActiveX is limited to the Windows platform.

6.1.1 What is OLE?

Object Linking and Embedding (OLE) is a Microsoft standard that allows you to integrate and reuse different software components within a single application.

Integrating a Developer/2000 application with a Microsoft Excel document, for example, enables you to offer both Developer/2000 and Microsoft Excel features. Your users can format a Microsoft Excel document with any of the text processing features provided by Microsoft Excel, while using Developer/2000 features for displaying and manipulating data from the database.

By incorporating OLE objects within your application, you can seamlessly integrate a diverse group of specialized components within Developer/2000 to build full-fledged applications. You no longer have to build entire applications from the ground up. You can deliver applications in a shorter amount of time and at a lower cost.
6.1.1 When should I use OLE within Developer/2000?

Use OLE when:

- You want to leverage an existing OLE-compliant application within your Developer/2000 application.
  
  For example, you can enhance your application’s capabilities with word processor documents, spreadsheet documents, knob controls, video clips, sound, and so on.

- You want to provide your application users with a familiar interface.
  
  On Microsoft Windows, most users are familiar with Microsoft Word and Microsoft Excel. Rather than creating word processing or spreadsheet functionality to your application, you could leverage and embed a Word or Excel document within your Developer/2000 application.

- Your applications are primarily deployed on the Windows platform.

6.1.1.2 About OLE servers and containers

OLE uses the concept of client and server. The client is an application that requests and uses the services of another application. The server is the one that provides these services.

- **OLE Server Application**
  
  An OLE server application creates OLE objects that are embedded or linked in OLE containers. The server application is responsible for the creation, storage, and manipulation of OLE objects. For example, the server decides how to repaint the object when certain portions are exposed.

  Graphics Builder and Microsoft Word are examples of OLE servers.

- **OLE Container Application**
  
  Unlike OLE server applications, OLE container applications do not create documents for embedding and linking. Instead, OLE container applications provide a place to store and display objects that are created by OLE server applications.

  Form Builder and Report Builder are examples of OLE container applications.
6.1.1.3 About embedded and linked objects
You can link or embed OLE objects within your application.

- **Embedded Object.** An embedded object has both its presentation and native data stored within your Developer/2000 application, or as a LONG RAW column in the database.

- **Linked Object.** A linked object only contains presentation information and a reference to its native data. The content of the linked object is not stored within your application or as a LONG RAW column in a database; it is stored in a separate, linked file.

There is no functional difference between linking and embedding. The OLE container treats the objects equally, by executing the same code, whether they are linked or embedded. The only difference is that embedding an OLE object increases the size of your application. This could eventually lead to performance considerations (particularly on a file server), because the larger the application, the longer it will take Developer/2000 to open and load it into memory.

6.1.1.4 About the registration database
Each client machine contains an OLE registration database. The registration database stores a set of classes that categorize OLE objects. The information in the registration database determines the object classes that are available for embedding and linking in OLE containers.

OLE server applications export a set of classes that become members of the registration database. Each computer has a single registration database. If the registration database does not already exist when an OLE server application is installed, one is created.

A single OLE server application can add many OLE classes to the registration database. The process of adding classes to the registration database is transparent and occurs during the installation of an OLE server application. For example, when you install Microsoft Excel, several classes are added to the registration database; some of the classes that are installed in the registration database include Excel Application, Excel Application 5, Excel Chart, Excel Sheet, ExcelMacrosheet, and ExcelWorkSheet.
6.1.1.5 About OLE activation styles

Activating an OLE object enables you to access features from the OLE server application. There are two ways to activate an OLE object: in-place activation or external activation.

- **In-place Activation.** In-place activation enables your users to manipulate the OLE object within your Developer/2000 application without switching to a different window.

  During in-place activation, the activated object appears within a hatched border, and the toolbar, menu and other controls of the activated object temporarily replace Developer/2000’s menu options. The replacement menu options and toolbars provide access to features that are available from the OLE server application. Developer/2000 menu options and toolbars re-appear when you deactivate in-place activation. To deactivate in-place activation, you click anywhere outside the hatched border.

  **Note:** In-place activation is available for embedded objects, but it is not available for linked objects.

- **External Activation.** External activation enables your users to manipulate the OLE object in a separate window. When an OLE object is activated, the object’s OLE server application is launched, and the OLE object appears in a separate OLE server application window. The separate window has the menu options and toolbars of the OLE server application. To deactivate external activation, you must explicitly exit the OLE server application.

  External activation is available for both embedded and linked objects.

  When the contents of a linked source file is modified with external activation, a linked object can be updated manually or automatically. Manual updates require an explicit instruction for an object to reflect changes from a linked source file. Automatic updates occur as soon as you modify a linked source file.

  **Note:** Both in-place activation and external activation are dependent on the OLE activation property settings of the OLE container. If the OLE server application is accessible, the activation property settings of the OLE container determine whether in-place activation or external activation occurs when an embedded OLE object is activated. Linked objects can only be activated with external activation; in-place activation does not apply to linked objects, even if the in-place activation property is set to Yes.
### 6.1.1.6 About OLE automation

Occasionally, you may want to interact with or manipulate the data within an OLE object. To do so, you use PL/SQL and OLE automation.

OLE automation enables the server application to expose a set of commands and functions that can be invoked from an OLE container application. By using these commands and functions, you can manipulate OLE objects from the OLE container environment.

In Developer/2000, you use PL/SQL to access any command or function that is exposed by an OLE server application. Developer/2000 built-ins provide a PL/SQL Application Programming Interface for creating, manipulating, and accessing OLE commands and functions.

**Note:** Many of the options available for manipulating an OLE object in an OLE container application are determined by the OLE server application. For instance, options from the OLE popup menu, also known as OLE verbs, are exposed by the OLE server application. The information contained in the registration database, such as object classes, is also dependent on the OLE server application.

### 6.1.1.7 OLE support in Developer/2000

Developer/2000 provides OLE server and container support. Developer/2000 also supports OLE automation.

<table>
<thead>
<tr>
<th>Developer/2000</th>
<th>Container</th>
<th>Server Application</th>
<th>OLE2 Automation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Form Builder</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Graphics Builder</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Procedure Builder</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Project Builder</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Query Builder</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Report Builder</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Schema Builder</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Translation Builder</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
6.1.1.7.1 **OLE container support**

As OLE container applications, Form Builder and Report Builder support the following:

- Embedding and linking of OLE server objects into OLE containers.
- In-place activation of embedded contents in OLE containers (Form Builder only).
  
  In-place activation enables you to access menus and toolbars from OLE server applications to edit embedded OLE objects while you are in Form Builder.
- Programmatic access to OLE objects, properties, and methods through OLE automation support from PL/SQL.
  
  Using PL/SQL, you can invoke commands and functions that are exposed by OLE servers.
- Seamless storage of OLE objects in a database in LONG RAW columns.
  
  You can save OLE objects to a database, as well as query OLE objects from a database. When linked objects are saved, only the image and the link information are retained in the database. The contents of a linked object remains in a linked source file. Saving an embedded object retains all the contents of an embedded object in the database.

6.1.1.7.2 **OLE server support**

Graphics Builder is an OLE server application. You can embed or link Graphics Builder displays within your Developer/2000 application.

**Recommendation:** If you want to add a Graphics Builder display to your application, don’t embed or link it as an OLE object. Instead, use Developer/2000’s Chart Wizard to add graphical displays to your applications.
6.1.1.7.3 OLE container properties

OLE container properties determine OLE display attributes, OLE container interaction with the server, container storage, and so on.

**Note:** In addition to container properties, you can also set OLE object properties. Each OLE object can expose several properties. You access OLE object properties by clicking the right mouse button to display the popup menu.

This section lists the OLE container properties supported by Developer/2000.

<table>
<thead>
<tr>
<th>Developer/2000</th>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Form Builder</td>
<td>OLE Activation Style</td>
<td>Specifies the event that will activate the OLE containing item, either double-click, focus-in, or manual.</td>
</tr>
<tr>
<td></td>
<td>OLE Class</td>
<td>Determines what class of OLE objects can reside in an OLE container.</td>
</tr>
<tr>
<td></td>
<td>OLE In-place Activation</td>
<td>Specifies if OLE in-place activation is used for editing embedded OLE objects.</td>
</tr>
<tr>
<td></td>
<td>OLE Inside-Out Support</td>
<td>Specifies if the OLE server of the embedded object enables inside-out object support during in-place activation. Inside-out activation enables for more than one embedded object to have an active editing window within an OLE container.</td>
</tr>
<tr>
<td></td>
<td>OLE Popup Menu Items</td>
<td>Determines which OLE popup menu commands are displayed and enabled when the mouse cursor is on the OLE object and the right mouse button is pressed. The OLE popup menu commands manipulate OLE objects.</td>
</tr>
<tr>
<td></td>
<td>OLE Resize Style</td>
<td>Determines how an OLE object is displayed in an OLE container.</td>
</tr>
</tbody>
</table>
### 6.1 Working with OLE Objects and ActiveX Controls

<table>
<thead>
<tr>
<th>Developer/2000 Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLE Tenant Aspect</td>
<td>Determines how an OLE object appears in an OLE container, either content, icon, or thumbnail.</td>
</tr>
<tr>
<td>OLE Tenant Types</td>
<td>Specifies the type of OLE objects that can be tenants of the OLE container, either embedded, linked, any, static, or none.</td>
</tr>
<tr>
<td>Show OLE Popup Menu</td>
<td>Determines whether the right mouse button displays a popup menu of commands for interacting with the OLE object.</td>
</tr>
<tr>
<td>Show OLE Tenant Type</td>
<td>Determines whether a border defining the OLE object type surrounds the OLE container.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Report Builder</th>
<th>Create New</th>
<th>Specifies that you want to embed your OLE object within your report application.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Create from File</td>
<td>Specifies that you want to link your OLE object within your report application.</td>
</tr>
<tr>
<td></td>
<td>Display as Icon</td>
<td>Specifies whether the OLE object should appear as an icon. By default, the OLE object appears as an empty container.</td>
</tr>
</tbody>
</table>
### 6.1.1.7.4 OLE/ActiveX built-ins

This section lists the OLE and ActiveX built-ins supported by Developer/2000.

<table>
<thead>
<tr>
<th>Developer/2000</th>
<th>Built-in</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Form Builder</td>
<td>ACTIVATE_SERVER</td>
<td>Activates an OLE server associated with an OLE container and prepares the OLE server to receive OLE automation events from the OLE container.</td>
</tr>
<tr>
<td></td>
<td>ADD_OLEARGS</td>
<td>Establishes the type and value of an argument that will be passed to the OLE object’s method.</td>
</tr>
<tr>
<td></td>
<td>CALL_OLE</td>
<td>Passes control to the identified OLE object’s method.</td>
</tr>
<tr>
<td></td>
<td>CALL_OLE_&lt;return type&gt;</td>
<td>Passes control to the identified OLE object’s method. Receives a return value of the specified type. There are four versions of the function (denoted by the value in returntype), one for each of the argument types CHAR, NUM, OBJ, and VAR.</td>
</tr>
<tr>
<td></td>
<td>CLOSE_SERVER</td>
<td>Deactivates the OLE server associated with an OLE container. Terminates the connection between an OLE server and the OLE container.</td>
</tr>
<tr>
<td></td>
<td>CREATE_OLEOBJ</td>
<td>In its first form, creates an OLE object, and establishes the object’s persistence. In its second form, alters the persistence of a previously-instantiated OLE object.</td>
</tr>
<tr>
<td></td>
<td>CREATE_VAR</td>
<td>Creates an empty, unnamed variant. There are two versions of the function, one for scalars and the other for arrays.</td>
</tr>
</tbody>
</table>
6.1 Working with OLE Objects and ActiveX Controls

<table>
<thead>
<tr>
<th>Developer/2000</th>
<th>Built-in</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>■ DESTROY_VARIANT</td>
<td></td>
<td>Destroys a variant that was created by the CREATE_VAR function.</td>
</tr>
<tr>
<td>■ EXEC_VERB</td>
<td></td>
<td>Causes the OLE server to execute the verb identified by the verb name or the verb index. An OLE verb specifies the action that you can perform on an OLE object.</td>
</tr>
<tr>
<td>■ FIND_OLE_VERB</td>
<td></td>
<td>Returns an OLE verb index. An OLE verb specifies the action that you can perform on an OLE object, and each OLE verb has a corresponding OLE verb index.</td>
</tr>
<tr>
<td>■ GET_INTERFACE_POINTER</td>
<td></td>
<td>Returns a handle to an OLE2 automation object.</td>
</tr>
<tr>
<td>■ GET_OLEARG_&lt;type&gt;</td>
<td></td>
<td>Obtains the nth argument from the OLE argument stack. There are four versions of the function (denoted by the value in type), one for each of the argument types CHAR, NUM, OBJ, and VAR.</td>
</tr>
<tr>
<td>■ GET_OLE_MEMBERID</td>
<td></td>
<td>Obtains the member ID of a named method or property of an OLE object.</td>
</tr>
<tr>
<td>■ GET_VAR_BOUNDS</td>
<td></td>
<td>Obtains the bounds of an OLE variant’s array.</td>
</tr>
<tr>
<td>■ GET_VAR_DIMS</td>
<td></td>
<td>Determines if an OLE variant is an array, and if so, obtains the number of dimensions in that array.</td>
</tr>
<tr>
<td>■ GET_VAR_TYPE</td>
<td></td>
<td>Obtains the type of an OLE variant.</td>
</tr>
</tbody>
</table>
### Developer/2000 Built-in Description

<table>
<thead>
<tr>
<th>Developer/2000</th>
<th>Built-in</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>■ GET_VERB_COUNT</td>
<td>Returns the number of verbs that an OLE server recognizes. An OLE verb specifies the action that you can perform on an OLE object, and the number of verbs available depends on the OLE server.</td>
<td></td>
</tr>
<tr>
<td>■ GET_VERB_NAME</td>
<td>Returns the name of the verb that is associated with the given verb index.</td>
<td></td>
</tr>
<tr>
<td>■ INITIALIZE_CONTAINER</td>
<td>Inserts an OLE object from a server-compatible file into an OLE container.</td>
<td></td>
</tr>
<tr>
<td>■ INIT_OLE_ARGS</td>
<td>Establishes how many arguments are going to be defined and passed to the OLE object's method.</td>
<td></td>
</tr>
<tr>
<td>■ LAST_OLE_ERROR</td>
<td>Returns the identifying number of the most recent OLE error condition.</td>
<td></td>
</tr>
<tr>
<td>■ LAST_OLE_EXCEPTION</td>
<td>Returns the identifying number of the most recent OLE exception that occurred in the called object.</td>
<td></td>
</tr>
<tr>
<td>■ OLEVAR_EMPTY</td>
<td>An OLE variant of type VT_EMPTY.</td>
<td></td>
</tr>
<tr>
<td>■ PTR_TO_VAR</td>
<td>First, creates an OLE variant of type VT_PTR that contains the supplied address. Then, passes that variant and type through the function VARPTR_TO_VAR.</td>
<td></td>
</tr>
<tr>
<td>■ RELEASE_OBJ</td>
<td>Shuts down the connection to the OLE object.</td>
<td></td>
</tr>
<tr>
<td>■ SERVER_ACTIVE</td>
<td>Indicates whether or not the server associated with a given container is running.</td>
<td></td>
</tr>
</tbody>
</table>
### 6.1 Working with OLE Objects and ActiveX Controls

<table>
<thead>
<tr>
<th>Developer/2000 OLE2 Package</th>
<th>Built-in</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SET_OLE</td>
<td>Changes the value of an OLE property. There are three versions of the procedure, one for each of the new-value types: NUMBER, VARCHAR, and OLEVAR.</td>
</tr>
<tr>
<td></td>
<td>SET_VAR</td>
<td>Sets a newly-created OLE variant to its initial value. Or, resets an existing OLE variant to a new value. There are four versions of the procedure, one for each of the new value types CHAR, NUMBER, OLEVAR, and table.</td>
</tr>
<tr>
<td></td>
<td>TABLE_FROM_BLOCK</td>
<td>Populates a table from a block.</td>
</tr>
<tr>
<td></td>
<td>TO_VARIANT</td>
<td>Creates an OLE variant and assigns it a value. There are four versions of the function.</td>
</tr>
<tr>
<td></td>
<td>VARPTR_TO_VAR</td>
<td>Changes a variant pointer into a simple variant.</td>
</tr>
<tr>
<td></td>
<td>VAR_TO_TABLE</td>
<td>Reads an OLE array variant and populates a PL/SQL table from it.</td>
</tr>
<tr>
<td></td>
<td>VAR_TO_&lt;type&gt;</td>
<td>Reads an OLE variant and transforms its value into an equivalent PL/SQL type. There are three versions of the function (denoted by the value in type), one for each for of the types CHAR, NUM and OBJ.</td>
</tr>
<tr>
<td></td>
<td>VAR_TO_VARPTR</td>
<td>Creates an OLE variant that points to an existing variant.</td>
</tr>
<tr>
<td></td>
<td>ADD_ARG</td>
<td>Adds an argument to a given argument list.</td>
</tr>
<tr>
<td></td>
<td>CREATE_ARGLIST</td>
<td>Creates an argument list to be passed to an OLE server.</td>
</tr>
</tbody>
</table>
# 6.1 Working with OLE Objects and ActiveX Controls

<table>
<thead>
<tr>
<th>Developer/2000</th>
<th>Built-in</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>■ CREATE_OBJ</td>
<td></td>
<td>Returns a handle to a newly created OLE object. This is usually used for OLE objects that do not have a user interface, such as a spell-checker.</td>
</tr>
<tr>
<td>■ DESTROY_ARGLIST</td>
<td></td>
<td>Destroys the specified argument list.</td>
</tr>
<tr>
<td>■ GET_CHARPROPERTY</td>
<td></td>
<td>Returns a character property of the OLE object.</td>
</tr>
<tr>
<td>■ GET_NUMPROPERTY</td>
<td></td>
<td>Returns a number property of the OLE object.</td>
</tr>
<tr>
<td>■ GET_OBJPROPERTY</td>
<td></td>
<td>Returns an object type property of the OLE object.</td>
</tr>
<tr>
<td>■ INVOKE</td>
<td></td>
<td>Executes the specified OLE server procedure.</td>
</tr>
<tr>
<td>■ INVOKE_CHAR</td>
<td></td>
<td>Executes the specified OLE server function. This function returns a character value.</td>
</tr>
<tr>
<td>■ INVOKE_NUM</td>
<td></td>
<td>Executes the specified OLE server function. This function returns a number value.</td>
</tr>
<tr>
<td>■ INVOKE_OBJ</td>
<td></td>
<td>Executes the specified OLE server function. This function returns an object type value.</td>
</tr>
<tr>
<td>■ LAST_EXCEPTION</td>
<td></td>
<td>Returns an OLE error.</td>
</tr>
<tr>
<td>■ SET_PROPERTY</td>
<td></td>
<td>Sets the OLE property with the specified value.</td>
</tr>
<tr>
<td>■ RELEASE_OBJ</td>
<td></td>
<td>Deallocates all resources for the specified OLE object.</td>
</tr>
</tbody>
</table>
### 6.1.8 OLE guidelines

When working with OLE objects, consider these guidelines:

<table>
<thead>
<tr>
<th>Item</th>
<th>Recommendation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Embedding or Linking an OLE object</td>
<td>You should link an OLE object when:</td>
</tr>
<tr>
<td></td>
<td>■ Your users prefer to work with the OLE object within the OLE server environment (your users prefer external activation). You link your OLE object when your users are more comfortable editing a spreadsheet, for example, within Microsoft Excel, rather than within your Developer/2000 application.</td>
</tr>
<tr>
<td></td>
<td>■ The OLE object is used in multiple applications.</td>
</tr>
<tr>
<td></td>
<td>■ The size of your application is a concern.</td>
</tr>
<tr>
<td></td>
<td>You should embed an OLE object when:</td>
</tr>
<tr>
<td></td>
<td>■ Your users can work with OLE objects within your Developer/2000 application; your users prefer in-place activation.</td>
</tr>
<tr>
<td></td>
<td>■ You prefer to maintain a single application, rather than maintaining an application with multiple OLE source files.</td>
</tr>
<tr>
<td></td>
<td>■ You are not concerned about the size of your application.</td>
</tr>
<tr>
<td>OLE Activation Style</td>
<td>You should use external activation. Linked objects can only be activated with external activation.</td>
</tr>
<tr>
<td>Display Style</td>
<td>For optimum performance, set the Display Style property for your OLE object to Icon.</td>
</tr>
<tr>
<td>Creating OLE objects at design-time or runtime?</td>
<td>You should create your OLE objects at design-time.</td>
</tr>
<tr>
<td></td>
<td>When you create an OLE container within Form Builder, Developer/2000 automatically initializes the OLE object.</td>
</tr>
<tr>
<td></td>
<td>In contrast, if you insert an OLE object at runtime, you must initialize the OLE object manually.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> If you manually insert an OLE object during Forms Runtime, the OLE object appears in the OLE container until the next record query. For any subsequent record queries, the OLE container appears in a state as defined in the Form Builder or populated with an OLE object from the database.</td>
</tr>
</tbody>
</table>
### 6.1.1.9 Adding an OLE object to your application

For detailed steps about how to add an OLE object to your application, refer to the online help.

### 6.1.1.10 Manipulating OLE objects

OLE server applications expose a set of commands that allow you to manipulate an OLE object programmatically.

You can manipulate OLE objects by:

- Getting and setting OLE properties.
- Calling OLE methods to perform special commands.

**Note:** Before you can call an OLE method, you must first import the OLE object’s methods and properties into Developer/2000. Importing OLE methods and properties enables you to interact with the OLE object within the native Developer/2000 environment.

You can access OLE methods from your application by using the `STANDARD` (Form Builder only) and `OLE2` built-in packages.

---

<table>
<thead>
<tr>
<th>Item</th>
<th>Recommendation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Portability</td>
<td>OLE objects are only supported on Microsoft Windows. If portability is an issue, you should not incorporate OLE objects within your application. Instead, consider developing the features within Developer/2000, or consider developing a 3GL foreign function.</td>
</tr>
<tr>
<td>Setting OLE properties within Report Builder</td>
<td>Report Builder OLE container properties are only available in the Create OLE Object dialog; Report Builder does not expose OLE container properties within the Property Palette. When working within Report Builder, set OLE properties within the Create OLE Object dialog.</td>
</tr>
</tbody>
</table>
6.1.11 OLE examples

This section provides several examples to help you get started with OLE.

6.1.11.1 Example 1: setting an OLE property using bind variable syntax

Within your form applications, you can use the
: item('item_name').ocx.server_name.property bind variable syntax to assign or retrieve property values.

For example:

```
: item('OLEitem').OCX.SpreadSheet.CellForeGroundColor:=
: item('OLEitem').OCX.SpreadSheet.CellForeGroundColor + 1;
```

OLEitem is the name of the item, SpreadSheet is the name of the OLE control server, and CellForeGroundColor is the name of the property.

6.1.11.2 Example 2: setting an OLE property using property assessors

Within your form applications, you can also use property assessor functions and procedures to get and set property values.

For example:

```
Variant OleVar;
Variant := EXCEL_WORKSHEET.ole_Range(:CTRL.interface,
To_variant('A1'));
```

EXCEL_WORKSHEET is the name of the program unit created from the OLE Importer. OLE_RANGE is the name of the property accessor.

6.1.11.3 Example 3: modifying cells in an Excel spreadsheet

This example gets and sets cell values in an Excel spreadsheet.

```
PACKAGE spreadsheet IS
   procedure setcell(trow number, col number, val number);
   function getcell(trow number, col number) return number;
END;

PACKAGE BODY spreadsheet IS
   obj_hnd ole2.obj_type; /* store the object handle */
   FUNCTION get_object_handle return ole2.obj_type IS
      BEGIN
         /* If the server is not active, activate the server and get the object handle. */
      END;
```

```
if not forms_ole.server_active ('spreadsheet') then
forms_ole.activate_server ('spreadsheet');
obj_hnd := forms_ole.get_interface_pointer ('spreadsheet');
end if;
return obj_hnd;
END;
/*
Excel cells are accessed with the following syntax in Visual Basic:
ActiveSheet.Cells(row, column).Value
In PL/SQL, we need to first get the active sheet using the
forms_ole.get_interface_pointer built-in. We can then use that to call the
Cells method with the row and column in an argument list to get a handle to
that specific cell. Lastly, we access the value of that cell.
*/

PROCEDURE SETCELL (trow number, col number, val number) IS
  d ole2.obj_type;
  c ole2.obj_type;
  n number;
  lst ole2.list_type;
BEGIN
  /* Activate the server and get the object handle
to the spreadsheet.
  */
  d := get_object_handle;
  /* Create an argument list and insert the specified
row and column into the argument list.
  */
  lst := ole2.create_arglist;
  ole2.add_arg(lst,trow);
  ole2.add_arg(lst,col);
  /* Call the Cells method to get a handle to the
specified cell.
  */
  c := ole2.invoke_obj(d,'Cells',lst);
  /* Set the value of that cell. */
  ole2.set_property(c,'Value',val);
  /* Destroy the argument list and the cell object
handle.
  */
  ole2.destroy_arglist(lst);
  ole2.release_obj(c);
END;
FUNCTION GETCELL(trow number, col number) return number IS
  c ole2.obj_type;
  d ole2.obj_type;
  n number;
  lst ole2.list_type;
BEGIN
  /* Activate the server and get the object handle
     to the spreadsheet. */
  d := get_object_handle;
  /* Create an argument list and insert the specified
     row and column into the argument list. */
  lst := ole2.create_arglist;
  ole2.add_arg(lst,trow);
  ole2.add_arg(lst,col);
  /* Call the Cells method to get the value in the
     specified cell. */
  c := ole2.invoke_obj (d,'Cells',lst);
  n := ole2.get_num_property (c, 'Value');
  /* Destroy the argument list. */
  ole2.destroy_arglist(lst);
  ole2.release_obj(c);
  return n;
END;
END;

To access a cell, use the following code:

spreadsheet.set_cell(3, 5, 91.73);
:block1.item1 := spreadsheet.get_cell(2, 4);
6.1.2 What are ActiveX controls?

ActiveX controls (originally known as OLE or OCX controls) are stand-alone software components that you embed within your application to provide light-weight user interface controls.

ActiveX controls differ from OLE objects in several ways:

- An ActiveX control is not a separate application, but a server that plugs into an ActiveX container—ActiveX controls are self-contained.
- Each ActiveX control exposes a set of properties, methods, and events. Properties define the ActiveX control’s physical and logical attributes, methods define actions that the ActiveX control can perform, and events denote some change in status in the ActiveX control.
- ActiveX controls must be deployed and installed with your applications.

6.1.2.1 When should I use ActiveX controls within Developer/2000?

ActiveX controls are typically used to enhance an application by providing some additional, self-contained functionality.

For example, you can enhance your application with a tabbed property sheet, a spin control, a calendar control, a help control, and so on.

A significant amount of effort is required to develop your own ActiveX controls or OLE servers. It is recommended that you use ActiveX controls and OLE servers developed and distributed by third party vendors.

6.1.2.2 Manipulating ActiveX controls

Each ActiveX control exposes a set of properties, methods, and events. Properties define the ActiveX control’s physical and logical attributes, methods define actions that the ActiveX control can perform, and events denote some change in status in the ActiveX control.

You can manipulate an ActiveX control by:

- Setting and getting ActiveX control properties.
- Calling ActiveX control methods.

Note: Before you can invoke an ActiveX control method, you must first import its methods and events into Developer/2000. Importing ActiveX methods and events enables you to interact with the ActiveX control within the native Developer/2000 environment.
6.1 Working with OLE Objects and ActiveX Controls

- Responding to ActiveX control events.

To manipulate an ActiveX control, you use the STANDARD (Form Builder) and OLE2 (Developer/2000) built-in packages.

6.1.2.3 Responding to ActiveX events

You can respond to an ActiveX event by writing your own code within an ActiveX event package or within the On-Dispatch-Event trigger.

Each ActiveX event is associated with a PL/SQL procedure defined in the events’ package. When the control fires an event, the code in the procedure is automatically executed.

Procedure names are determined by an internal number that represents the corresponding event. The restricted procedure produced by an event has an application programming interface similar to the following:

```
PROCEDURE /*Click*/ event4294966696(interface OleObj);
```

Note: ActiveX procedures run in restricted mode. When calling the event procedure within an On-Dispatch-Event trigger, you can explicitly define whether the procedure is run in restricted or unrestricted mode by using the FORMS4W.DISPATCH_EVENT call. When defining a restricted procedure, OUT parameters are not observed.

6.1.2.4 Deploying your ActiveX control

Deploying an application that contains an ActiveX control requires that you deploy the ActiveX control.

To deploy an ActiveX control, you must:

- Register the ActiveX control on the client-machine.
  
  If you install an ActiveX control by using the installation program supplied with the ActiveX control, registration occurs automatically.
  
  For manual registration, use regActiveX32.exe or regsvr32.exe; both are available with Microsoft development tools and from ActiveX control vendors.

- Copy ActiveX DLLs to the client-machine (for example, C:\WINDOWS\SYSTEM).
  
  Most ActiveX controls require a supporting DLL, such as the Microsoft Foundation Class runtime library (MFC40.DLL). The DLL must be in the \WIN-
DOWS\SYSTEM directory or in the search path. If the DLL is out of date or missing, your ActiveX control will not register properly.

Note: ActiveX controls, whether distributed by third party ActiveX control vendors or bundled with application development tools, may require that you pay additional fees or obtain additional licenses prior to distributing the ActiveX control.

6.1.2.5 ActiveX support in Developer/2000
Developer/2000 provides complete support for creating, manipulating, and communicating with ActiveX controls.

<table>
<thead>
<tr>
<th>Developer/2000</th>
<th>Container</th>
</tr>
</thead>
<tbody>
<tr>
<td>Form Builder</td>
<td>Yes</td>
</tr>
<tr>
<td>Graphics Builder</td>
<td>No</td>
</tr>
<tr>
<td>Procedure Builder</td>
<td>No</td>
</tr>
<tr>
<td>Project Builder</td>
<td>No</td>
</tr>
<tr>
<td>Query Builder</td>
<td>No</td>
</tr>
<tr>
<td>Report Builder</td>
<td>No</td>
</tr>
<tr>
<td>Schema Builder</td>
<td>No</td>
</tr>
<tr>
<td>Translation Builder</td>
<td>No</td>
</tr>
</tbody>
</table>

6.1.2.5.1 ActiveX properties
This section lists the ActiveX properties supported by Developer/2000.

<table>
<thead>
<tr>
<th>Developer/2000</th>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Form Builder</td>
<td>OLE Class</td>
<td>Determines what class of OLE objects can reside in an OLE container.</td>
</tr>
</tbody>
</table>
6.1 Working with OLE Objects and ActiveX Controls

6.1.2.5.2 ActiveX/OLE built-ins

Refer to Section 6.1.1.7.4 for a list of the ActiveX and OLE built-ins supported by Developer/2000.

6.1.2.6 ActiveX guidelines

This section provides guidelines for working with ActiveX controls.

<table>
<thead>
<tr>
<th>Developer/2000 Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control Properties</td>
<td>Allows you to set ActiveX control properties. You can access the ActiveX properties dialog through the Property Palette or by clicking the ActiveX control, then clicking the right mouse button.</td>
</tr>
<tr>
<td>About Control</td>
<td>Displays information about the ActiveX control</td>
</tr>
<tr>
<td>Control Help</td>
<td>Displays control-specific help (if available).</td>
</tr>
</tbody>
</table>

### 6.1.2.5.2 ActiveX/OLE built-ins

Refer to Section 6.1.1.7.4 for a list of the ActiveX and OLE built-ins supported by Developer/2000.

### 6.1.2.6 ActiveX guidelines

This section provides guidelines for working with ActiveX controls.

<table>
<thead>
<tr>
<th>Item</th>
<th>Recommendation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creating your own ActiveX Control</td>
<td>A significant amount of effort is required to develop your own ActiveX controls or OLE servers. It is recommended that you use ActiveX controls and OLE servers developed and distributed by third party vendors.</td>
</tr>
<tr>
<td>Initializing an ActiveX Control</td>
<td>Use ActiveX controls in blocks with the Single Record property set to Yes, because single records are immediately initialized when Forms Runtime starts up. For multiple records, each record is not initialized until you navigate to the record. Without initialization, the ActiveX control item is empty, giving the impression that no ActiveX control is available.</td>
</tr>
</tbody>
</table>
### Item Recommendation

<table>
<thead>
<tr>
<th>Item</th>
<th>Recommendation</th>
</tr>
</thead>
</table>
| Managing OLE Variant Types | - Some OLE servers such as Microsoft Excel use variant types. Use the `STANDARD` built-in package to do the necessary conversion to and from variant types.  

- The lifetime and scope of a variant type is limited to a trigger (variant memory space is released when a trigger exits). To extend the lifetime and scope of a variant type, set the persistent parameter in `To_Variant()` to `TRUE` and assign the results to a global variable.  

**Note:** Global variants must be explicitly destroyed using `Destroy_Variant()`. Similarly, OLE objects created with `Create_OleObj()` are global in scope (the persistent parameter defaults to `TRUE`). You must explicitly call `Release_Obj()` to release global objects.|
| Moving ActiveX Files | You should maintain your ActiveX files within the “install” directory; do not move your ActiveX files to a different directory.  

At installation, the directories in which the ActiveX control is installed are registered in the Windows Registration Database in Windows 95 and Windows NT, making the ActiveX Control visible to your development environment.  

When you move an ActiveX Control to a different directory, or rename the directory, you invalidate the information in the registry.  

If you find it necessary to move the ActiveX Control or rename its directory, use `regsvr32.exe` or `regActiveX.exe` utilities provided with most Microsoft development products to re-register the ActiveX in its new location. |
| Portability Issues | Developer/2000 supports ActiveX on the Windows platform only. ActiveX controls cannot be used on the Web or on UNIX. If portability is an issue, do not use an ActiveX control. |
Debugging ActiveX Calls

Given that object types cannot be checked at compile time, it is possible to call a function on an object which is not defined for the class of that object. Because the functions are bound by ID rather than by name, a different function may be called than expected, leading to unusual errors.

One way to guarantee that you are calling the correct method is to change the generated function, replacing the hardcoded constant with a call to GET_OLE_MEMBERID. For example:

```sql
Procedure Ole_Add(interface OleObj, TimeBegin VARCHAR2, TimeEnd VARCHAR2, Text VARCHAR2, BackColor OleVar := OleVar_Null) IS
  BEGIN
    Init_OleArgs (4);
    Add_OleArg (TimeBegin);
    Add_OleArg (TimeEnd);
    Add_OleArg (Text);
    Add_OleArg (BackColor);
    Call_Ole (interface, 2);
  END

Replace the Call_Ole() with:
  Call_Ole (interface, Get_Ole_MemberID(interface, 'Add'));
You can check for FORM_SUCCESS after the GET_OLE_MEMBERID call.
```
6.1 Working with OLE Objects and ActiveX Controls

6.1.2.7 Adding an ActiveX control to your application
For information about how to add an ActiveX control to your Developer/2000 application, refer to the online help.

6.1.2.8 ActiveX examples
This section provides several examples to help you get started with ActiveX controls.

6.1.2.8.1 Example 1: setting ActiveX control properties
In Form Builder, you can use the :item('item_name').ocx.server_name.property bind variable syntax to assign or retrieve ActiveX property values.

For example:

: item ('ActXitem').OCX.Spindial.spindialctrl1.1.Needleposition :=
: item ('ActXitem').OCX.Spindial.spindialctrl1.1.Needleposition + 1;

ActXitem is the name of the item, Spindial.spindialctrl1.1 is the name of the ActiveX control server, and Needleposition is the name of the property.

The following code also works if your system.cursor_item is an ActiveX control:

: form.cursor_item.OCX.spindial.spindialctrl1.1.Needleposition :=
: form.cursor_item.OCX.spindial.spindialctrl1.1.Needleposition + 1;

Restrictions

- ActiveX event procedures are restricted. In general, GO_ITEM cannot be called within ActiveX procedure code, unless the same event applies to multiple items and a GO_ITEM is necessary. In this case, you can use the GO_ITEM built-in by doing the following: in the On-Dispatch-Trigger (block or form level), call DISPATCH_EVENT(RESTRICTED_ALLOWED). Note: You do not have to explicitly call the event procedure because it will automatically be called following the On-Dispatch trigger code.

- Initialization events for ActiveX controls do not fire in Forms Runtime. These initialization events are intentionally disabled. Instead, you can use When-New-Item-Instance or When-New-Record-Instance in place of the control’s native initialization events.

Item Recommendation
6.1.2.8.2 Example 2: getting ActiveX control properties

In Form Builder, you can use the property accessor functions and procedures to get and set ActiveX properties.

For example:

```sql
  tblname varchar2;
  tblname := table_pkg.TableName(item('Oblk.Oitm').interface);
```

`Table_pkg` is the name of the program unit created from the OLE Importer.
`TableName` is the name of the property accessor. `Oblk` is the name of the block and `Oitm` is the name of the item.
6.1.2.8.3 Example 3: calling ActiveX control methods

This example gets a cell value from a Spread Table ActiveX control by using the `GetCellByColRow` method, which is provided in the SpreadTable package.

```
DECLARE
    Cur_Row number;
    Cur_Col number;
    OLEObj OleObj;
BEGIN
    Cur_Row:=SpreadTable.CurrentRow(:ITEM('BLK.ITM').interface);
    Cur_Col:=SpreadTable.CurrentCol(:ITEM('BLK.ITM').interface);
    OLEObj:=SpreadTable.GetCellByColRow(:ITEM('BLK.ITM').interface, Cur_Col, Cur_Row);
END;
```
6.2 Using Foreign Functions to Customize Your Applications

Developer/2000 allows you to customize and supplement your applications with foreign functions.

This section addresses:

- Section 6.2.1, “What are foreign functions?”
- Section 6.2.2, “The foreign function interface”
- Section 6.2.3, “Foreign function guidelines”
- Section 6.2.4, “Creating a foreign function”
- Section 6.2.5, “Foreign function examples”

6.2.1 What are foreign functions?

Foreign functions are subprograms written in a 3GL programming language that allow you to customize your applications to meet the unique requirements of your users.

Foreign functions can interact with Oracle databases, and Developer/2000 variables, items, columns, and parameters. You can also call any external function, such as Windows DLLs or APIs.

6.2.1.1 When should I use a foreign function?

Foreign functions are often used to perform the following tasks:

- Perform complex data manipulation.
- Pass data to Developer/2000 from operating system text files.
- Manipulate LONG RAW data.
- Pass entire PL/SQL blocks for processing by the server.
- Set font and color attributes for Developer/2000 applications.
- Send mail directly from an application.
- Display Windows help as part of your application.
- Access the Microsoft Windows SDK.
- Leverage low-level system services, such as pipes.
- Control real time devices, such as a printer or a robot.
6.2 Using Foreign Functions to Customize Your Applications

6.2.1.2 Foreign function types
You can develop three types of foreign functions:

6.2.1.2.1 Oracle Precompiler foreign functions An Oracle Precompiler foreign function is the most common foreign function. Using the Oracle Precompiler, you can create foreign functions that access Oracle databases as well as Developer/2000 variables, items, columns, and parameters.

An Oracle Precompiler foreign function incorporates the Oracle Precompiler interface. This interface enables you to write a subprogram in one of the following supported host languages with embedded SQL commands: Ada, C, COBOL, FORTRAN, Pascal, and PL/I.

An Oracle Precompiler foreign function source file includes host programming language statements and Oracle Precompiler statements with embedded SQL statements. Precompiling an Oracle Precompiler foreign function replaces the embedded SQL statements with equivalent host programming language statements. After precompiling, you have a source file that you can compile with a host language compiler.

6.2.1.2.2 Oracle Call Interface (OCI) foreign functions An OCI foreign function incorporates the Oracle Call Interface. This interface enables you to write a subprogram that contains calls to Oracle databases. A foreign function that incorporates only the OCI (and not the Oracle Precompiler interface) cannot access Developer/2000 variables, items, columns, and parameters.

Note: You can also develop foreign functions that combine both the ORACLE Precompiler interface and the OCI.

6.2.1.2.3 Non-Oracle foreign functions A non-Oracle foreign function does not incorporate either the Oracle Precompiler interface or the OCI. For example, a non-Oracle foreign function might be written entirely in the C language. A non-Oracle foreign function cannot access Oracle databases, or Developer/2000 variables, items, columns, and parameters.
6.2 Using Foreign Functions to Customize Your Applications

6.2.2 The foreign function interface

Developer/2000 uses PL/SQL as its programming language. In order to call a foreign function, such as a C function in a Windows DLL, PL/SQL must have an interface to communicate with the foreign function.

Developer/2000 enables you to communicate with your foreign function through two distinct interfaces, either the Oracle Foreign Function Interface (ORA_FFI) or the user exit interface.

6.2.2.0.4 The Oracle Foreign Function Interface (ORA_FFI)

ORA_FFI is a portable and generic mechanism for enabling Developer/2000 to call 3GL routines from PL/SQL subprograms.

Foreign functions that are invoked from a PL/SQL interface must be contained in a dynamic library. Examples of dynamic libraries include dynamic link libraries on Microsoft Windows and shared libraries on UNIX systems.

6.2.2.0.5 User exit interface to foreign functions

The user exit interface is a platform-specific mechanism for enabling Developer/2000 to call 3GL routines from PL/SQL subprograms.

The foreign functions that you invoke from a user exit interface must be contained in a dynamic link library (.DLL) or linked with a Developer/2000 executable.

6.2.2.0.6 Comparing ORA_FFI and user exits

This section describes the advantages and disadvantages of using ORA_FFI and user exits.

<table>
<thead>
<tr>
<th>Foreign Function</th>
<th>Advantage</th>
<th>Disadvantage</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Exit</td>
<td>■ User exits are linked to an executable. This “tight binding” allows you to use and take advantage of the current database connection. ■ The most significant disadvantage to using user exits is the maintenance burden. You must relink your user exit whenever you modify your user exit or upgrade Developer/2000. ■ User exits are not generic; they are platform-specific.</td>
<td></td>
</tr>
</tbody>
</table>
6.2 Using Foreign Functions to Customize Your Applications

6.2.3 Foreign function guidelines

This section provides guidelines for working with foreign functions.

<table>
<thead>
<tr>
<th>Foreign Function</th>
<th>Advantage</th>
<th>Disadvantage</th>
</tr>
</thead>
</table>
| ORA_FFI          | - ORA_FFI is a pure PL/SQL specification. The ORA_FFI specification exists within a library (.PLL file), not within a Developer/2000 component. When you upgrade to a higher version of Developer/2000 or modify the foreign function, you don’t have to modify or regenerate the PLL file.  
- ORA_FFI is generic.  
- Developer/2000 provides several ORA_FFI packages (D2KWUTIL.PLL) that allow you to access libraries that are already available (Windows API functions). | - If you are using ORA_FFI and you are writing your own external code modules with Pro*C, you cannot use the current open database connection. You must open a second connection.  
- You cannot pass complex datatypes, such as structures or arrays. For example, you cannot use EXEC TOOLS GET or EXEC TOOLS PUT to interface with Developer/2000. |

### Item Recommendation

<table>
<thead>
<tr>
<th>Item</th>
<th>Recommendation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Which foreign function interface should I use?</td>
<td>Use the Oracle Foreign Function Interface (ORA_FFI). ORA_FFI is a portable, generic, and requires only minor or no maintenance</td>
</tr>
<tr>
<td>Can I perform screen I/O from a foreign function?</td>
<td>You should not perform host language screen I/O from a foreign function. This restriction exists because the runtime routines that a host language uses to perform screen I/O conflict with the routines that Developer/2000 uses to perform screen I/O. However, you can perform host language file I/O from a foreign function.</td>
</tr>
</tbody>
</table>
| Which host language should I use to write my user exit? | Your host language is a matter of preference. However, C is the recommended language.  
**Note:** Some C runtime functions are not available in .DLL files. For more information, refer to your compiler documentation. |
6.2 Using Foreign Functions to Customize Your Applications

For more information about foreign functions, refer to the following publications:

<table>
<thead>
<tr>
<th>Item</th>
<th>Recommendation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Which precompiler should I use to precompile my user exit?</td>
<td>You should use Pro*C version 2.2.3.0. When precompiling, be sure to specify the following MSVC compiler flags: Large, Segment Setup: SS != DS, DSloads on function entry Assume 'extern' and Uninitialized Data ‘far’ is checked Yes In Windows Prolog/Epilogue, Generate prolog/Epilogue for None</td>
</tr>
<tr>
<td>Do I have to recompile my user exit when I upgrade from a previous version of Developer/2000?</td>
<td>Yes. User exits can create a maintenance burden especially if you maintain several different executables, each with a different set of user exits. When you modify a user exit or upgrade to a higher version of Developer/2000, you must relink the user exit with Developer/2000 executables.</td>
</tr>
<tr>
<td>Can I deploy a foreign function on the Web?</td>
<td>ORA_FFI and user exits do not function on the Web. On web deployments, foreign functions interface with the DLLs on the server-side, not on the browser-side.</td>
</tr>
</tbody>
</table>

For more information about foreign functions, refer to the following publications:

<table>
<thead>
<tr>
<th>ORACLE Precompiler interface</th>
<th>Programmer's Guide to the ORACLE Precompilers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supported host languages</td>
<td>Developer/2000 Installation Guide for your operating system</td>
</tr>
<tr>
<td>Operating system-specific requirements when working with foreign functions</td>
<td>Developer/2000 online help</td>
</tr>
<tr>
<td>OCI</td>
<td>Oracle Call Interface Programmer’s Guide</td>
</tr>
<tr>
<td>Building DLLs</td>
<td>Developer/2000 online help system and your compiler documentation.</td>
</tr>
<tr>
<td>ORA_FFI</td>
<td>Developer/2000 online help</td>
</tr>
<tr>
<td>User Exits</td>
<td>Developer/2000 online help</td>
</tr>
<tr>
<td>PL/SQL</td>
<td>PL/SQL User’s Guide or the Developer/2000 online help</td>
</tr>
</tbody>
</table>
6.2.4 Creating a foreign function

This section provides detailed steps that describe how to create a foreign function interface:

- Creating an ORA_FFI interface to a foreign function
- Creating a user exit interface to a foreign function

6.2.4.1 Creating an ORA_FFI interface to a foreign function

The following example creates a PL/SQL package called WinSample. The WinSample package includes interfaces to the foreign function GetPrivateProfileString in the dynamic library KRNL386.EXE.

Note: When you create an ORA_FFI interface to a foreign function, you perform two basic steps. First, you create and associate a subprogram with a foreign function (the dispatcher function). By associating a PL/SQL subprogram with a foreign function, you can invoke the foreign function each time you call the associated PL/SQL subprogram. Associating a foreign function with a PL/SQL subprogram is necessary because Developer/2000 uses PL/SQL constructs. Second, you create a PL/SQL function which passes the arguments to the dispatcher function. The dispatcher function invokes the foreign function.

1. Create a package specification.

   Your package spec must represent the library. It must also define the PL/SQL function that you want to invoke.

   For example:

   ```plsql
   PACKAGE WinSample IS
   FUNCTION GetPrivateProfileString
     (Section IN VARCHAR2,
      Entry IN VARCHAR2,
      DefaultStr IN VARCHAR2,
      ReturnBuf IN OUT VARCHAR2,
      BufLen IN PLS_INTEGER,
      Filename IN VARCHAR2)
   RETURN PLS_INTEGER;
   END;
   ```

   In this example, you call the WinSample.GetPrivateProfileString PL/SQL function to invoke the GetPrivateProfileString foreign function in the dynamic library KRNL386.EXE.
6.2 Using Foreign Functions to Customize Your Applications

Note: You should check the parameters for the C function `GetPrivateProfileString`, and specify the matching PL/SQL parameter types and the PL/SQL return types. The C datatype `int` is equivalent to the PL/SQL parameter `IN PLS_INTEGER` and the PL/SQL return type `PLS_INTEGER`. The C datatype `char` is equivalent to the PL/SQL parameter `IN VARCHAR2`.

2. Define the library and its function handles.

For example:

```sql
PACKAGE BODY WinSample IS
  lh_KRNL386 ORA_FFI.LIBHANDLETYPE;
  fh_GetPrivateProfileString ORA_FFI.FUNCHANDLETYPE;

In this step, you declare the handle types for the library and the function. Later you will load the library and register the function using `ORA_FFI.LOAD_LIBRARY` and `ORA_FFI.REGISTER_FUNCTION`. Each of these functions returns a handle (a pointer) to the specified library and the function. `ORA_FFI.LIBHANDLETYPE` and `ORA_FFI.FUNCHANDLETYPE` are the PL/SQL datatypes for these handles.

3. Create the dispatcher function. The dispatcher function invokes your foreign function.

For example:

```sql
FUNCTION i_GetPrivateProfileString
  (funcHandle IN ORA_FFI.FUNCHANDLETYPE,
   Section IN OUT VARCHAR2,
   Entry IN OUT VARCHAR2,
   DefaultStr IN OUT VARCHAR2,
   ReturnBuf IN OUT VARCHAR2,
   BufLen IN PLS_INTEGER,
   Filename IN OUT VARCHAR2)
RETURN PLS_INTEGER;
PRAGMA INTERFACE(C,i_GetPrivateProfileString,11265);
```

The first argument of the dispatcher function that calls a foreign function must have at least one parameter, and the first parameter must be a handle to the registered foreign function that the subprogram invokes.

When you call the dispatcher function from the PL/SQL function, you pass the function handle as defined in step 2 (fh_GetPrivateProfileString).
When the dispatcher function gets called, the \texttt{PRAGMA} statement passes control to a memory location (11265 as specified in the above code) that communicates with the dynamic library.

4. Create the PL/SQL function that calls the dispatcher function. This PL/SQL function is the function that you defined in the package spec (Step 1).

For example:

\begin{verbatim}
FUNCTION GetPrivateProfileString
    (Section IN VARCHAR2,
    Entry IN VARCHAR2,
    DefaultStr IN VARCHAR2,
    ReturnBuf IN OUT VARCHAR2,
    BufLen IN PLS_INTEGER,
    Filename IN VARCHAR2)
RETURN PLS_INTEGER IS
    Section_l VARCHAR2(512)    := Section;
    Entry_l VARCHAR2(512)    := Entry;
    DefaultStr_l VARCHAR2(512) := DefaultStr;
    ReturnBuf_l VARCHAR2(512)  := RPAD(SUBSTR(NVL(ReturnBuf,' '),1,512),512,CHR(0));
    BufLen_l PLS_INTEGER    := BufLen;
    Filename_l VARCHAR2(512)   := Filename;
    rc PLS_INTEGER;
BEGIN
    rc  := i_GetPrivateProfileString
     (fh_GetPrivateProfileString,
    Section_l,
    Entry_l,
    DefaultStr_l, Ret
    urnBuf_l,
    BufLen_l, Filename_l);
    ReturnBuf := ReturnBuf_l;
    RETURN (rc);
END;
\end{verbatim}

This is the PL/SQL function you call from your Developer/2000 application. This function passes the arguments to the dispatcher function \texttt{i\_GetPrivateProfileString}, then \texttt{i\_GetPrivateProfileString} invokes the C function \texttt{GetPrivateProfileString} in \texttt{KRNL386.EXE}.

Recall that the first argument of a dispatcher function must be a function handle. Here \texttt{fh\_GetPrivateProfileString} is used to pass the function handle declared in Step 2.
5. Build the package body.

The package body must perform four steps to initialize a foreign function:

- Load the library
- Register the functions that are in the library
- Register the parameters (if any)
- Register the return type (if any)

For example:

```plaintext
BEGIN
/* Load the library.*/
lh_KRNL386 := ORA_FFI.LOAD_LIBRARY
('location of the DLL here','KRNL386.EXE');

/* Register the foreign function. */
fh_GetPrivateProfileString := ORA_FFI.REGISTER_FUNCTION
(lh_KRNL386,'GetPrivateProfileString',ORA_FFI.PASCAL_STD);

/* Register the parameters. */
ORA_FFI.REGISTER_PARAMETER
(fh_GetPrivateProfileString,ORA_FFI.C_CHAR_PTR);
ORA_FFI.REGISTER_PARAMETER
(fh_GetPrivateProfileString,ORA_FFI.C_CHAR_PTR);
ORA_FFI.REGISTER_PARAMETER
(fh_GetPrivateProfileString,ORA_FFI.C_CHAR_PTR);
ORA_FFI.REGISTER_PARAMETER
(fh_GetPrivateProfileString,ORA_FFI.C_CHAR_PTR);
ORA_FFI.REGISTER_PARAMETER
(fh_GetPrivateProfileString,ORA_FFI.C_CHAR_PTR);

/* Register the return type. */
ORA_FFI.REGISTER_RETURN(fh_GetPrivateProfileString,ORA_FFI.C_INT);
END WinSample;
```
Recall that you declared two handles for the library and the function in Step 2. In this step, you assign values to the handles by using the
ORA_FFI.LOAD_LIBRARY and ORA_FFI.REGISTER_FUNCTION functions.

ORA_FFI.LOAD_LIBRARY takes two arguments: the location and the name of
the dynamic library. ORA_FFI.REGISTER_FUNCTION takes three arguments:
the library handle for the library where the function resides, the function name,
and the calling standard. The calling standard can be either C_STD (for the C
calling standard) or PASCAL_STD (for the Pascal calling standard).

After you load the library and register the function, you must register the
parameters and return types (if there are any).

ORA_FFI.REGISTER_PARAMETER and ORA_FFI.REGISTER_RETURN take
two arguments each: the function handle and the argument type.

6. Within Developer/2000, create a library file (.PLL) that includes your package,
then attach it to your application.

7. Call the foreign function from your application.

For example:

```pascal
x := Winsample.GetPrivateProfileString
('Oracle', 'ORACLE_HOME', '<Not Set>', 'Value', 100, 'oracle.ini');
```
6.2.4.2 Creating a user exit interface to a foreign function

User exits are not generic; they are platform-specific. Some details of implementing user exits are specific to each operating system. The following example describes how to create a user exit on Windows 95.

On Microsoft Windows, a foreign function that can be invoked from a user exit is contained in a dynamic link library (.DLL). A DLL is a library that loads into memory only when the contained code is invoked.

6.2.4.2.1 Example: creating a user exit on Windows 95

The following example creates a foreign function that adds an ID column to the EMP table.

This example uses several sample files, including:

- **UE_SAMP.MAK** is a project file that includes the IAPXTB control structure. Building this project generates **UE_SAMP.DLL**.
- **F50XTB32.DLL** is the default file containing foreign functions that can be invoked from a user exit interface. This file is a DLL that ships with Form Builder, and does not initially contain user-defined foreign functions. This file is placed in the **ORACLE_HOME\BIN** directory during installation. When you create new foreign functions, replace the existing **F50XTB32.DLL** file with a new **F50XTB32.DLL**.
- **UE_XTB.C** is a template source file for creating an IAPXTB control structure. **UE_XTB.C** contains an example of an entry for the IAPXTB control structure. Modify this file and add your foreign function entries.
- **UE.H** is a sample header file that is used to define the IAPXTB control structure.
- **F50XTB.DEF** contains definitions you need to build your own DLL. Use **F50XTB.DEF** to export foreign functions. **F50XTB.DEF** contains several export statements. You should not modify these export statements as they are used by Form Builder to access the user exit interface.
- **OSSWEP.OBJ** is the Dynamic Link Library Windows Entry Point .OBJ file you need to build your own DLL. (You may replace **OSSWEP.OBJ** with an .OBJ file of your own.)
- **UEZ.OBJ** is an .OBJ file that you link to your own .OBJ files.
The user exit sample files are located in your ORACLE_HOME directory (for example, C:\ORAWIN95\FORMS50\USEREXIT).

1. Write a foreign function.
   For example, create a text file called UEXIT_PC, then add the following:
   ```c
   /* UEXIT_PC file */
   /* This foreign function adds an ID column to the EMP table. */
   ifndef UE
   #include "ue.h"
   endif
   ifndef _WINDLL
   #define SQLCA_STORAGE_CLASS extern
   endif
   EXEC SQL INCLUDE sqlca.h
   void AddColumn()
   {
      EXEC SQL alter table EMP add ID varchar(9);
   }
   ```

2. Precompile the foreign function with the Pro*C precompiler.
   For example, use Pro*C to precompile the UEXIT_PC file. When you precompile UEXIT_PC, Pro*C creates a C file called UEXIT_C.
   **Note:** When precompiling, be sure to specify the following MSVC compiler flags:
   - Large, Segment Setup: SS != DS, DSloads on function entry
   - Assume ‘extern’ and Uninitialized Data ‘far’ is checked Yes
   - In Windows Prolog/Epilogue, Generate prolog/Epilogue for None

3. Create your header files.
   Your header file must define your foreign function.
   For example, modify the sample header file, UE_H, by adding the following:
   ```c
   extern void AddColumn();
   ```

4. Create the IAPXTB control structure.
   For example, modify the sample file, UE_XTB.C, by adding an include statement for UE_H (# include "ue.h"), the name of the user exit (Add_ID_Column), the name of the foreign function (AddColumn), and the language type(XITCC).
6.2 Using Foreign Functions to Customize Your Applications

```c
#ifndef UE
#include "ue.h"
#endif /* UE */
#include "ue_samp.h"

/* Define the user exit table */
exit iapxtb[] = { /* Holds exit routine pointers */
   "Add_ID_Column", AddColumn, XITCC,
   (char *) 0, 0, 0 /* zero entry marks the end */
}; /* end iapxtb */
```

5. Build your DLL. The steps for building a DLL vary depending on your particular compiler. For more information, refer to your compiler documentation.

For example, using your compiler, create a project that contains:
UE_SAMP.MAK, F50XTB.DEF, OSSWRP.OBJ, UEZ.OBJ, UE_XTB.C, and UEEXIT.C.

Before building your DLL, you must link the following files:
LIBC.LIB
OLDNAMES
C:\ORAWIN95\FORMS50\USEREXIT\F50R.LIB
C:\ORAWIN95\PRO20\USEREXIT\SQL18WIN.LIB
C:\ORAWIN95\PRO20\USEREXIT\SQX18WIN.LIB

After building the UE_SAMP.MAK project, the result is a DLL named
UE_SAMP.DLL. Add the UE_SAMP.DLL entry to the list of DLLs defined by
the FORM50_USEREXITS parameter in the registry.

Alternatively, you can rename UE_SAMP.DLL to F50XTB32.DLL, backup the
F50XTB32.DLL in the C:\ORAWIN95\BIN directory, and copy the new
F50XTB32.DLL to the C:\ORAWIN95\BIN directory.

6. Invoke the foreign function from a user exit.

For example, create a When-Button-Pressed Trigger that calls the foreign function from a user exit.

The following statement demonstrates how to invoke the AddColumn foreign function by specifying the user exit name Add_ID_Column in the USER_EXIT built-in:

```c
/* Trigger: When-Button-Pressed */
USER_EXIT('Add_ID_Column');
```
6.2.5 Foreign function examples

This section includes several examples that describe how to use foreign functions.

**Note:** These examples are available online; refer to the “Foreign Function examples” topic in the Form Builder online help.

### 6.2.5.1 Using ORA_FFI to call Windows help

```c
/* WinHelp ORA_FFI. */
/* */
/* */
/* Usage: WinHelp.WinHelp(helpfile VARCHAR2, */
/* command  VARCHAR2, */
/* data    {VARCHAR2/PLS_INTEGER See Below}) */
/* */
/* command can be one of the following: */
/* */
/* 'HELP_INDEX'         Help Contents */
/* 'HELP_CONTENTS'    " */
/* 'HELP_CONTEXT'       Context Key (See below) */
/* 'HELP_KEY'           Key Search */
/* 'HELP_PARTIALKEY'    Partial Key Search */
/* 'HELP_QUIT'          Quit */
/* */
/* data contains a string for the key search or a numeric context */
/* value if using topics. */
/* */
/* Winhelp.Winhelp('C:\ORAWIN95\TOOLS\DOC20\OF50HUS.HLP', */
/* 'HELP_PARTIALKEY', */
/* 'ORA_FFI'); */
/* */
/* The commented sections replace the line below if using HELP_CONTEXT keys */
```

```sql
PACKAGE WinHelp IS
    FUNCTION WinHelp(helpfile IN VARCHAR2, 
                      command IN VARCHAR2, 
                      data IN VARCHAR2) 
        RETURN PLS_INTEGER;
END;
```

```sql
PACKAGE BODY WinHelp IS
    lh_USER ora_ffi.libHandleType;
    fh_WinHelp ora_ffi.funcHandleType;

    FUNCTION i_WinHelp(funcHandle IN ora_ffi.funcHandleType,
```
6.2 Using Foreign Functions to Customize Your Applications

```sql
hwnd       IN PLS_INTEGER,
helpfile   IN OUT VARCHAR2,
command    IN PLS_INTEGER,
data       IN OUT VARCHAR2)
RETURN PLS_INTEGER;
PRAGMA INTERFACE(C,i_WinHelp,11265);

FUNCTION WinHelp(helpfile IN VARCHAR2,
command  IN VARCHAR2,
data     IN VARCHAR2)
RETURN PLS_INTEGER
IS
hwnd_l     PLS_INTEGER;
helpfile_l VARCHAR2(512) := helpfile;
command_l  PLS_INTEGER;
data_l     VARCHAR2(512) := data;
rc         PLS_INTEGER;

FUNCTION Help_Convert(command IN VARCHAR2)
RETURN PLS_INTEGER
IS
BEGIN
/* The windows.h definitions for command */
/* HELP_CONTEXT      0x0001 */
/* HELP_QUIT         0x0002 */
/* HELP_INDEX        0x0003 */
/* HELP_CONTENTS     0x0003 */
/* HELP_HELPONHELP   0x0004 */
/* HELP_SETINDEX     0x0005 */
/* HELP_SETCONTENTS  0x0005 */
/* HELP_CONTEXTPOPUP 0x0008 */
/* HELP_FORCEFILE    0x0009 */
/* HELP_KEY          0x1001 */
/* HELP_COMMAND      0x1002 */
/* HELP_PARTIALKEY   0x1005 */
/* HELP_MULTIKEY     0x0201 */
/* HELP_SETWINPOS    0x0203 */

if command = 'HELP_CONTEXT' then return(1); end if;
if command = 'HELP_KEY' then return(257); end if;
if command = 'HELP_PARTIALKEY' then return(261); end if;
if command = 'HELP_QUIT' then return(2); end if;
/* If nothing else go to the contents page */
```

Taking Advantage of Developer/2000’s Open Architecture 6-277
6.2 Using Foreign Functions to Customize Your Applications

```
return (3);
END;

BEGIN
hwnd_l :=
TO_PLS_INTEGER(Get_Item_Property(name_in('SYSTEM.CURSOR_ITEM'),WINDOW_HANDLE));

command_l := Help_Convert(command);

rc := i_WinHelp(fh_WinHelp,
    hwnd_l,
    helpfile_l,
    command_l,
    data_l);

RETURN (rc);
END;

BEGIN
BEGIN
lh_USER := ora_ffi.find_library('USER.EXE');
EXCEPTION WHEN ora_ffi.FFI_ERROR THEN
    lh_USER := ora_ffi.load_library(NULL,'USER.EXE');
END;

fh_WinHelp :=
ora_ffi.register_function(lh_USER,'WinHelp',ora_ffi.PASCAL_STD);

ora_ffi.register_parameter(fh_WinHelp,ORA_FFI.C_INT); /*(HWND)*/
ora_ffi.register_parameter(fh_WinHelp,ORA_FFI.C_CHAR_PTR); /*LPCSTR*/
ora_ffi.register_parameter(fh_WinHelp,ORA_FFI.C_INT); /*UINT*/
ora_ffi.register_parameter(fh_WinHelp,ORA_FFI.C_CHAR_PTR); /*DWORD*/
ora_ffi.register_return(fh_WinHelp,ORA_FFI.C_INT); /*BOOL*/

END WinHelp;

6.2.5.2 Using ORA_FFI to open the File Open dialog on Windows

PACKAGE OraDlg IS
FUNCTION OraMultiFileDlg
(Title IN VARCHAR2,
Filter IN VARCHAR2,
```
6.2 Using Foreign Functions to Customize Your Applications

Dir IN VARCHAR2,
FileStream IN OUT VARCHAR2)
RETURN PLS_INTEGER;
FUNCTION OraSingleFileDlg
(Title IN VARCHAR2,
Filter IN VARCHAR2,
Dir IN VARCHAR2,
FileStream IN OUT VARCHAR2)
RETURN PLS_INTEGER;
END OraDlg;
PACKAGE BODY OraDlg IS
   lh_ORADLG ora_ffi.libHandleType;
   fh_OraMultiFileDlg ora_ffi.funcHandleType;
   fh_OraSingleFileDlg ora_ffi.funcHandleType;
   FUNCTION i_OraMultiFileDlg
   (funcHandle IN ora_ffi.funcHandleType,
    Title IN OUT VARCHAR2,
    Filter IN OUT VARCHAR2,
    Dir IN OUT VARCHAR2,
    FileStream IN OUT VARCHAR2)
   RETURN PLS_INTEGER;
   PRAGMA INTERFACE(C,i_OraMultiFileDlg,11265);
   FUNCTION OraMultiFileDlg
   (Title IN VARCHAR2,
    Filter IN VARCHAR2,
    Dir IN VARCHAR2,
    FileStream IN OUT VARCHAR2)
   RETURN PLS_INTEGER;
   PRAGMA INTERFACE(C,OraMultiFileDlg,11265);
   FUNCTION OraSingleFileDlg
   (Title IN VARCHAR2,
    Filter IN VARCHAR2,
    Dir IN VARCHAR2,
    FileStream IN OUT VARCHAR2)
   RETURN PLS_INTEGER IS
    Title_l VARCHAR2(128) := RPAD(SUBSTR(NVL(Title,'Open'),1,128),128,CHR(0));
    Filter_l VARCHAR2(128) := RPAD(SUBSTR(NVL(Filter,'All Files (*.*)|*.*|'),1,128),128,CHR(0));
    Dir_l VARCHAR2(256) := RPAD(SUBSTR(NVL(Dir,' '),1,256),256,CHR(0));
    FileStream_l VARCHAR2(2000) := RPAD(SUBSTR(NVL(FileString,''),1,2000),2000,CHR(0));
    rc PLS_INTEGER;
BEGIN
    rc := i_OraMultiFileDlg(fh_OraMultiFileDlg,
    Title_l,
    Filter_l,
    Dir_l,
    FileStream_l);
    FileStream := FileStream_l;
RETURN (rc);
END;
FUNCTION i_OraSingleFileDlg
6.2 Using Foreign Functions to Customize Your Applications

```sql
(funcHandle IN ora_ffi.funcHandleType,
Title IN OUT VARCHAR2,
Filter IN OUT VARCHAR2,
Dir IN OUT VARCHAR2,
FileString IN OUT VARCHAR2)
RETURN PLS_INTEGER;
PRAGMA INTERFACE(C,i_OraSingleFileDlg,11265);
FUNCTION OraSingleFileDlg
(Title IN VARCHAR2,
Filter IN VARCHAR2,
Dir IN VARCHAR2,
FileString IN OUT VARCHAR2)
RETURN PLS_INTEGER IS
Title_l VARCHAR2(128) := RPAD(SUBSTR(NVL(Title,'Open'),1,128),128,CHR(0));
Filter_l VARCHAR2(128) := RPAD(SUBSTR(NVL(Filter,'All Files (*.*)|*.*|'),1,128),128,CHR(0));
Dir_l VARCHAR2(256) := RPAD(SUBSTR(NVL(Dir,' '),1,256),256,CHR(0));
FileString_l VARCHAR2(2000) := RPAD(SUBSTR(NVL(FileString,''),1,2000),2000,CHR(0));
rc PLS_INTEGER;
BEGIN
rc := i_OraSingleFileDlg(fh_OraSingleFileDlg,
Title_l,
Filter_l,
Dir_l,
FileString_l);
FileString := FileString_l;
RETURN (rc);
END;
BEGIN
lh_ORADLG := ora_ffi.find_library('ORADLG.DLL');
EXCEPTION WHEN ora_ffi.FFI_ERROR THEN
lh_ORADLG := ora_ffi.load_library(NULL,'ORADLG.DLL');
END;
fh_OraMultiFileDlg := ora_ffi.register_function
(lh_ORADLG,'OraMultiFileDlg',ora_ffi.PASCAL_STD);
ora_ffi.register_parameter(fh_OraMultiFileDlg,ORA_FFI.C_CHAR_PTR);
ora_ffi.register_parameter(fh_OraMultiFileDlg,ORA_FFI.C_CHAR_PTR);
ora_ffi.register_parameter(fh_OraMultiFileDlg,ORA_FFI.C_CHAR_PTR);
ora_ffi.register_parameter(fh_OraMultiFileDlg,ORA_FFI.C_CHAR_PTR);
ora_ffi.register_return(fh_OraMultiFileDlg,ORA_FFI.C_LONG);
fh_OraSingleFileDlg := ora_ffi.register_function
(lh_ORADLG,'OraSingleFileDlg',ora_ffi.PASCAL_STD);
ora_ffi.register_parameter(fh_OraSingleFileDlg,ORA_FFI.C_CHAR_PTR);
```

6-280 Developer/2000: Guidelines for Building Applications
ora_ffi.register_parameter(fh_OraSingleFileDlg,ORA_FFI.C_CHAR_PTR);
ora_ffi.register_parameter(fh_OraSingleFileDlg,ORA_FFI.C_CHAR_PTR);
ora_ffi.register_parameter(fh_OraSingleFileDlg,ORA_FFI.C_CHAR_PTR);
ora_ffi.register_return(fh_OraSingleFileDlg,ORA_FFI.C_LONG);
END OraDlg;

6.2.5.3 Using ORA_FFI to call Unix(SUN) executables with a STDIN/STDOUT type interface

/* Copyright (c) 1997 by Oracle Corporation */
/
/*

NAME
ora_pipe_io_spec.sql - Specification for access to Unix Pipe mechanism

DESCRIPTION
Demonstration of how to use the ORA_FFI Package to provide access to the
Unix Pipe C functions.

PUBLIC FUNCTION(S)
popen    - Open the Pipe command
get_line - Get a line of Text from a Pipe
put_line - Put a line of Text into a Pipe
pclose   - Close the Pipe
is_open  - Determine whether the Pipe descriptor is open.

NOTES

In Order to use these routines you could write the following
PL/SQL Code:

-- Example of Calls to ora_pipe_io functions
DECLARE
    stream ora_pipe_io.PIPE;
    buffer VARCHAR2(240);
BEGIN
    stream := ora_pipe_io.popen('ls -l', ora_pipe_io.READ_MODE);

    loop
        exit when not ora_pipe_io.get_line(stream, buffer, 240);
        :directory.file := buffer;
        down;
    end loop;

    ora_pipe_io.pclose(stream);
END;

MODIFIED (MM/DD/YY)
smclark 08/05/94 - Creation
PACKAGE ora_pipe_io is

 /*
 ** Arguments to popen.
 */
 READ_MODE constant VARCHAR2(1) := 'r';
 WRITE_MODE constant VARCHAR2(1) := 'w';

 /* ------------ FUNCTION POPEN ----------- */
 ** Function POPEN -- Open a Un*x pipe command
 **
 ** Given a Unix command to execute and a Pipe read/write mode in which
 ** to execute the instruction this Function will execute the Command
 ** and return a handle, of type PIPE, to the resulting Input/Output
 ** stream.
 **
 ** The command to be executed is limited to 1024 characters.
 */
 FUNCTION popen(command in VARCHAR2,
 ctype in VARCHAR2)
 RETURN PIPE;

 /* ------------ PROCEDURE PCLOSE ----------- */
 ** Procedure PCLOSE -- Close a pipe
 **
 ** Close a previously opened pipe.
 **
 ** Raises a VALUE_ERROR exception if incorrect arguments are passed.
PROCEDURE pclose(stream in out PIPE);

FUNCTION get_line(stream in out PIPE, s in out VARCHAR2, n in PLS_INTEGER) RETURN BOOLEAN;

PROCEDURE put_line(stream in out PIPE, s in VARCHAR2);
6.2 Using Foreign Functions to Customize Your Applications

END;

/*  ora_pipe_io_body.sql - Body of Package for access to Unix Pipe mechanism
   DESCRIPTION
   Demonstration of how to use the ORA_FFI Package to provide access to the
   Unix Pipe C functions.
   PUBLIC FUNCTION(S)
   popen    - Open the Pipe command
   get_line - Get a line of Text from a Pipe
   put_line - Put a line of Text into a Pipe
   pclose   - Close the Pipe
   is_open  - Determine whether the Pipe descriptor is open.
   PRIVATE FUNCTION(S)
   icd_popen, icd_fgets, icd_fputs, icd_pclose
   NOTES
   MODIFIED   (MM/DD/YY)
   smclark    11/02/94 -  Modified for production release changes to ORA_FFI.
   smclark    08/05/94 -  Creation
*/

PACKAGE BODY ora_pipe_io is
  lh_libc   ora_ffi.libHandleType;
  fh_popen  ora_ffi.funcHandleType;
  fh_pclose ora_ffi.funcHandleType;
  fh_fgets  ora_ffi.funcHandleType;
  fh_fputs  ora_ffi.funcHandleType;

  /* ------------ FUNCTION ICD_POPEN ----------- */
  /*
   ** Function ICD_POPEN -- Interface routine to C function popen
   **
   ** This function acts as the interface to the popen function in
   ** libc.
   */
  FUNCTION icd_popen(funcHandle in ora_ffi.funcHandleType,
                      command in out VARCHAR2,
                      ctype in out VARCHAR2)
    return ORA_FFI.POINTERTYPE;
  pragma interface(c, icd_popen, 11265);

  /* ------------ PROCEDURE ICD_PCLOSE ----------- */
  /*
  ** Function ICD_PCLOSE -- Interface routine to C function pclose
  */
** This function acts as the interface to the pclose function in libc.

PROCEDURE icd_pclose(funcHandle in ora_ffi.funcHandleType,
                      stream in out ORA_FFI.POINTERTYPE);

pragma interface(c, icd_pclose, 11265);

/* ------------ FUNCTION ICD_FGETS ----------- */

/*
** Function ICD_FGETS -- Interface routine to C function fgets
**
** This function acts as the interface to the fgets function in libc.
*/

FUNCTION icd_fgets(funcHandle in ora_ffi.funcHandleType,
                     s in out VARCHAR2, n in PLS_INTEGER,
                     stream in out ORA_FFI.POINTERTYPE)
RETURN ORA_FFI.POINTERTYPE;

pragma interface(c, icd_fgets, 11265);

/* ------------ FUNCTION ICD_FPUTS ----------- */

/*
** Function ICD_FPUTS -- Interface routine to C function fputs
**
** This function acts as the interface to the fputs function in libc.
*/

PROCEDURE icd_fputs(funcHandle in ora_ffi.funcHandleType,
                     s in out VARCHAR2,
                     stream in out ORA_FFI.POINTERTYPE);

pragma interface(c, icd_fputs, 11265);

/* ------------ FUNCTION POPEN ----------- */

/*
** Function POPEN -- Open a Un*x pipe command
*/

FUNCTION popen(command in VARCHAR2,
                ctype in VARCHAR2)
RETURN PIPE is
    
    /*
     ** Take a copy of the arguments because we need to pass them
     ** IN OUT to icd_popen, but we really don't want people to have
     ** to call our routines in the same way.
     */
    cmd varchar2(1024) := command;
    cmode varchar2(1) := ctype;
    
    stream PIPE;
    BEGIN
        if (cmode not in (READ_MODE, WRITE_MODE))
            or (cmode is NULL)
            or (cmd is NULL)
        then
            raise VALUE_ERROR;
        end if;
        
        stream.file_handle := icd_popen(fh_popen, cmd, cmode);
        stream.is_open := TRUE;
        stream.read_write_mode := ctype;
        return(stream);
    END popen;
    
    /****************************************************************************************/
    /* PROCEDURE PCLOSE */
    /*
     ** Procedure PCLOSE -- Close a pipe
     */
    PROCEDURE pclose(stream in out PIPE) is
        BEGIN
            icd_pclose(fh_pclose, stream.file_handle);
            stream.is_open := FALSE;
        END pclose;
    /****************************************************************************************/
    /* FUNCTION GET_LINE */
    /*
     ** Function GET_LINE -- Get a line of text into a buffer
     ** from the read mode pipe.
     */
    FUNCTION get_line(stream in out PIPE,
        s in out VARCHAR2, n in PLS_INTEGER)
        RETURN BOOLEAN is
buffer ORA_FFI.POINTERTYPE;
BEGIN
  if (n <= 0)
    or (stream.is_open = FALSE)
    or (stream.is_open is NULL)
    or (stream.read_write_mode <> READ_MODE)
  then
    raise VALUE_ERROR;
  end if;

  /*
  ** Initialise the Buffer area to reserve the correct amount of space. 
  */
  s := rpad(' ', n);

  buffer := icd_fgets(fh_fgets, s, n, stream.file_handle);

  /*
  ** Determine whether a NULL pointer was returned.
  */
  return (ora_ffi.is_null_ptr(buffer) = FALSE);
END get_line;

PROCEDURE put_line(stream in out PIPE, 
s in VARCHAR2) is
  buffer varchar2(2048) := s;
BEGIN
  if (stream.is_open = FALSE)
    or (stream.is_open is NULL)
    or (stream.read_write_mode <> WRITE_MODE)
  then
    raise VALUE_ERROR;
  end if;

  icd_fputs(fh_fputs, buffer, stream.file_handle);
  buffer := chr(10);
  icd_fputs(fh_fputs, buffer, stream.file_handle);
END put_line;
/* ------------ FUNCTION IS_OPEN ----------- */
/*
** Function IS_OPEN -- Determines whether a pipe is open.
*/
FUNCTION is_open(stream in PIPE)
RETURN BOOLEAN is
BEGIN
   return(stream.is_open);
END is_open;
BEGIN
   /*
   ** Declare a library handle as libc.  (Internal so NULL,NULL)
   */
   lh_libc := ora_ffi.load_library(NULL, NULL);
   if ora_ffi.is_null_ptr(lh_libc) then
      raise VALUE_ERROR;
   end if;

   /*
   ** Register the popen function, it's return type and arguments.
   */
   fh_popen := ora_ffi.register_function(lh_libc, 'popen');
   if ora_ffi.is_null_ptr(fh_popen) then
      raise VALUE_ERROR;
   end if;
   ora_ffi.register_return(fh_popen, ORA_FFI.C_DVOID_PTR);
   ora_ffi.register_parameter(fh_popen, ORA_FFI.C_CHAR_PTR);
   ora_ffi.register_parameter(fh_popen, ORA_FFI.C_CHAR_PTR);

   /*
   ** Register the pclose function, it's return type and arguments.
   */
   fh_pclose := ora_ffi.register_function(lh_libc, 'pclose');
   if ora_ffi.is_null_ptr(fh_pclose) then
      raise VALUE_ERROR;
   end if;
   ora_ffi.register_return(fh_pclose, ORA_FFI.C_VOID);
   ora_ffi.register_parameter(fh_pclose, ORA_FFI.C_DVOID_PTR);

   /*
   ** Register the fgets function, it's return type and arguments.
   */
   fh_fgets := ora_ffi.register_function(lh_libc, 'fgets');
if ora ffi.is_null_ptr(fh_fgets) then
    raise VALUE_ERROR;
end if;
orar ffi.register_return(fh_fgets, ORA_FFI.C_DVOID_PTR);
orar ffi.register_parameter(fh_fgets, ORA_FFI.C_CHAR_PTR);
orar ffi.register_parameter(fh_fgets, ORA_FFI.C_INT);
orar ffi.register_parameter(fh_fgets, ORA_FFI.C_DVOID_PTR);
/*
 ** Register the fputs function, it's return type and arguments.
 */
fh_fputs := ora ffi.register_function(lib libc, 'fputs');
if ora ffi.is_null_ptr(fh_fputs) then
    raise VALUE_ERROR;
end if;
orar ffi.register_return(fh_fputs, ORA_FFI.C_VOID);
orar ffi.register_parameter(fh_fputs, ORA_FFI.C_CHAR_PTR);
orar ffi.register_parameter(fh_fputs, ORA_FFI.C_DVOID_PTR);
END ora pipe io;
6.3 Using the Open API to Build and Modify Form Builder Applications

This section describes the non-interactive, programmatic method for building and modifying Form Builder applications. It includes these topics:

- Section 6.3.1, “What is the Open API?”
- Section 6.3.2, “Guidelines for using the Open API”
- Section 6.3.3, “Using the Open API”
- Section 6.3.4, “Open API examples”

6.3.1 What is the Open API?

The Open API is an advanced Form Builder feature for C/C++ developers that want the power and flexibility to create or modify form modules in a non-interactive environment.

Note: Before using the Open API, you should have a thorough understanding of Form Builder objects and their properties and relations.

6.3.1.1 When should I use the Open API?

Use the Open API when you want to quickly propagate development changes to a large number of form modules. You might, for example, use the Open API to update your applications to the current corporate standards for look and feel. This could involve updating hundreds of form modules.

You can also use the Open API to:

- Compile a set of forms
- Collect dependency information
- Write your own documentation

6.3.1.2 Open API header files

The Open API consists of one C header file for each Form Builder object. There are 34 Form Builder objects (see the figure). These objects correspond to the Form Builder objects that you are familiar with at design-time. Each header file contains several functions and macros that you use to create and manipulate Form Builder-objects.
6.3 Using the Open API to Build and Modify Form Builder Applications
6.3.1.3 Open API properties
Within the Open API, you manipulate Form Builder objects by setting object properties.

Open API properties have their own unique names, such as `D2FP_FONT_NAM`. These properties correspond to the Form Builder properties that you are familiar with at design-time.

A property can be one of the following: Boolean, Text, Number, Object, or Blob.

The table below lists some common item properties with their corresponding Open API equivalents.

<table>
<thead>
<tr>
<th>Open API Property</th>
<th>Form Builder (design-time) Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>D2FP_ACCESS_KEY</td>
<td>Access Key</td>
</tr>
<tr>
<td>D2FP_BEVEL_STY</td>
<td>Bevel</td>
</tr>
<tr>
<td>D2FP_CNV_NAM</td>
<td>Canvas</td>
</tr>
<tr>
<td>D2FP_ENABLED</td>
<td>Enabled</td>
</tr>
<tr>
<td>D2FP_FONT_NAM</td>
<td>Font Name</td>
</tr>
<tr>
<td>D2FP_HEIGHT</td>
<td>Width/Height</td>
</tr>
<tr>
<td>D2FP_X_POS</td>
<td>X Position</td>
</tr>
<tr>
<td>D2FP_Y_POS</td>
<td>Y Position</td>
</tr>
</tbody>
</table>

6.3.1.4 Open API functions and macros
You use Open API functions and macros to create, destroy, duplicate, subclass, get, and set object properties.

For example, to determine an item’s font size, use the `D2FITMG_FONT_SIZ` macro:

```c
D2FITMG_FONT_SIZ(ctx, obj, val);
```

This macro returns the value of the Font Size property of the item object as type number.

To set a text item property, use the `D2FITMST_SETTEXTPROP` function:

```c
D2FITMST_SETTEXTPROP(d2fctx *pd2fctx, d2fitm *pd2fitm, ub2 pnum, text *prp);
```
6.3 Using the Open API to Build and Modify Form Builder Applications

This function sets the value of the specified item text property. You specify a pointer to the context in `pd2fctx`, the item in `pd2fitm`, the property number in `pnum`, and a handle to the text value in `prp`.

### 6.3.2 Guidelines for using the Open API

When working with the Open API, consider these guidelines:

<table>
<thead>
<tr>
<th>Item</th>
<th>Recommendation</th>
</tr>
</thead>
<tbody>
<tr>
<td>File Backups</td>
<td>The Open API is non-interactive; validation and error checking are not supported. Before using the Open API, you should backup your form modules (.FMBs).</td>
</tr>
</tbody>
</table>

Creating a relation object

- Create the object.
- Set relation object properties.
- Call the `d2frelup_Update` function to instantiate the object.

### 6.3.3 Using the Open API

This section provides detailed steps that describe how to create and modify Form Builder modules using the Open API.

#### 6.3.3.1 Creating and modifying modules using the Open API

To create or modify a Form Builder module:

1. Include the appropriate C header files in your C source code.
2. Make calls to the desired APIs in your C source code.
   - Initialize the context structure.
   - Make `load` function calls to open an existing form module, menu module, or object library.
   - Make the necessary Open Forms API function calls to perform the desired operations, including connecting to an existing database, if required.
   - Generate an .FMX or .MMX compiled form using the appropriate `CompileFile()` function.
6.3 Using the Open API to Build and Modify Form Builder Applications

- Make the required function calls to save the associated module (for example, d2ffmdsv_Save() for a form module, d2fmmdsv_Save() for a menu module, or d2folbsv_Save() for an object library).
- Finally, call the context destroy function, d2fctxde_Destroy(), to destroy the Open Forms API context. Note that this function call must be your final one.

3. Link your source files against the Open API library (f50d2f32.lib).
4. Compile the files to create an executable (.EXE file).
5. Run the executable to create or modify your Form modules (.FMB).

6.3.4 Open API Examples

This section includes several examples that describe how to use the Open API.

Note: These examples are available online; refer to the “Open API examples” topic in the Form Builder online help.

6.3.4.1 Modifying modules using the Open API

/*
This example determines if the Form Builder object is a subclassed object and returns the file path of the parent to NULL if the object is subclassed. This sample only processes the following object types: form level triggers, alerts, blocks, items, item level triggers, radio buttons, and block level triggers. Use a similar method to process other object types.
*/
#include <stdio.h>
#include <string.h>
#include <windows.h>
#include <d2ferr.h>
#include <d2fctx.h>
#include <d2ffmd.h>
#include <d2fblk.h>
#include <d2fitm.h>
#include <d2falt.h>
#include <d2ftrg.h>
#include <d2frdb.h>
#define BUFSIZE 128
int WINAPI WinMain(HANDLE hInstance,
  HANDLE hPrevInstance,
  LPSTR lpszCommandLine,
  int cmdShow)
{

6.3 Using the Open API to Build and Modify Form Builder Applications

```c
D2FCTX* pd2fctx;
D2FFMD* pd2ffmd;
D2FBLK* pd2fblk;
D2FITM* pd2fitm;
D2FCTX d2fctx_attr;
D2FSTATUS status;
D2FALT* pd2falt;
D2FTRG* pd2ftrg;
D2FRDB* pd2frdb;
int counter;
char buf[BUFSIZE];
char* form_name=(char*)0;
/* Get the form name from the command line */
strcpy(buf, lpszCommandLine, BUFSIZE);
form_name = strtok(buf, ".");
/* Initialize the attribute mask */
d2fctx_attr.mask_d2fctxa = 0;
/* for MS Windows-only attributes */
d2fctx_attr.d2fihnd_d2fctxa = hInstance;
d2fctx_attr.d2fphnd_d2fctxa = hPrevInstance;
d2fctx_attr.d2fcmsh_d2fctxa = cmdShow;
/* Create the API context */
status = d2fctxcr_Create(&pd2fctx, &d2fctx_attr);
/* Load the form */
status = d2ffmdld_Load(pd2fctx, &pd2ffmd, form_name, FALSE);
if (status == D2FS_D2FS_SUCCESS)
{
    /** Process Form Level Trigger Objects **/
    for(status = d2ffmdg_trigger(pd2fctx, pd2ffmd, &pd2ftrg);
        pd2ftrg != NULL;
        status = d2ftrgg_next(pd2fctx, pd2ftrg, &pd2ftrg))
    {
        if (d2ftrgis_IsSubclassed(pd2fctx, pd2ftrg) == D2FS_YES)
            d2ftrgs_par_flpath(pd2fctx, pd2ftrg, NULL);
    }
    /** Process Alert Objects **/
    for(status = d2ffmdg_alert(pd2fctx, pd2ffmd, &pd2falt);
        pd2falt != NULL;
        status = d2faltg_next(pd2fctx, pd2falt, &pd2falt))
    {
        if (d2faltis_IsSubclassed(pd2fctx, pd2falt) == D2FS_YES)
            d2falts_par_flpath(pd2fctx, pd2falt, NULL);
    }
    /** Process Block Objects **/
    for(status = d2ffmdg_block(pd2fctx, pd2ffmd, &pd2fblk);
```
6.3 Using the Open API to Build and Modify Form Builder Applications

```c
pd2fblk != NULL;
status = d2fblkg_next(pd2fctx,pd2fblk,&pd2fblk))
{
if (d2fblkis_IsSubclassed(pd2fctx,pd2fblk) == D2FS_YES)
   d2fblks_par_flpath(pd2fctx,pd2fblk,NULL);
}
/* Process Item Objects */
for(status = d2fblkg_item(pd2fctx,pd2fblk,&pd2fitm);
    pd2fitm != NULL;
    status = d2fitmg_next(pd2fctx,pd2fitm,&pd2fitm))
{
if (d2fitmis_IsSubclassed(pd2fctx,pd2fitm) == D2FS_YES)
   d2fitms_par_flpath(pd2fctx,pd2fitm,NULL);
/* Process Item Level Trigger Objects */
for(status = d2fitmg_trigger(pd2fctx,pd2fitm,&pd2ftrg);
    pd2ftrg != NULL;
    status = d2ftrgg_next(pd2fctx,pd2ftrg,&pd2ftrg))
{
if (d2ftrgis_IsSubclassed(pd2fctx,pd2ftrg)==D2FS_YES)
   {d2ftrgs_par_flpath(pd2fctx,pd2ftrg,NULL);
    printf("item trigger is Subclassed\n")
   }
   else if  (d2ftrgis_IsSubclassed(pd2fctx,
   pd2ftrg)==D2FS_NO)
    printf("item trigger is NOT Subclassed\n");
}
/* Process Radio Button Objects */
for(status = d2fitmg_rad_but(pd2fctx,pd2fitm,&pd2frdb);
    pd2frdb != NULL;
    status = d2frdbs_next(pd2fctx,pd2frdb,&pd2frdb))
{
if (d2frdbis_IsSubclassed(pd2fctx,pd2frdb)==D2FS_YES)
   {d2frdbs_par_flpath(pd2fctx,pd2frdb,NULL);
    printf("radio button is Subclassed\n");
   }
   else if (d2frdbis_IsSubclassed(pd2fctx,
   pd2frdb)==D2FS_NO)
    printf("radio button is NOT Subclassed\n");
}
/* Process Block Level Trigger Objects */
for(status = d2fblkg_trigger(pd2fctx,pd2fblk,&pd2ftrg);
    pd2ftrg != NULL;
    status = d2ftrgg_next(pd2fctx,pd2ftrg,&pd2ftrg))
{
if (d2ftrgis_IsSubclassed(pd2fctx,pd2ftrg)==D2FS_YES)
   {d2ftrgs_par_flpath(pd2fctx,pd2ftrg,NULL);
    printf("item trigger is Subclassed\n");
   }
   else if  (d2ftrgis_IsSubclassed(pd2fctx,
   pd2ftrg)==D2FS_NO)
    printf("item trigger is NOT Subclassed\n");
}
```

6-296  Developer/2000: Guidelines for Building Applications
6.3.4.2 Creating modules using the Open API

/*
This example creates a master-detail form based on the dept and emp database tables owned by the user scott. The master contains the following fields: empno, ename, job, sal, and deptno. The detail contains the following fields deptno, dname, and loc. The join condition is deptno.
*/

#include<stdio.h>
#include<string.h>
#include<windows.h>
#include<d2fctx.h>
#include<d2ffmd.h>
#include<d2ffpr.h>
#include<d2fob.h>
#include<d2fcnv.h>
#include<d2ftrg.h>
#include<d2blk.h>
#include<d2fitm.h>
#include<d2fwin.h>
#include<d2frel.h>
#define D2FS_SUCCESS 0
#define FAIL 1
#define BUFSIZE 128
6.3 Using the Open API to Build and Modify Form Builder Applications

```c
#define WBP_TXT "null;\n"
int WINAPI WinMain(HANDLE hInstance,
    HANDLE hPrevInstance,
    LPSTR lpszCommandLine,
    int cmdShow)
{
    d2fctx  *pd2fctx;
    d2ffmd  *pd2ffmd;
    d2fcnv  *pd2fcnv;
    d2fwin  *pd2fwin;
    d2fblk  *pempblk;
    d2fblk  *pdeptblk;
    d2fre1  *pd2fre1;
    d2fitm  *pEempnoitm;
    d2fitm  *pEenameitm;
    d2fitm  *pEjobitm;
    d2fitm  *pEsalitm;
    d2fitm  *pEdeptnoitm;
    d2fitm  *pDdeptnoitm;
    d2fitm  *pDdnameitm;
    d2fitm  *pDlocitm;
    text  *name = (text *)0;
    text  *form_name = (text *)0;
    d2fctxa  d2fctx_attr;
    d2fstatus retval;
    char buf[BUFSIZE];
    /* Get form name */
    strncpy(buf, "empdept", BUFSIZE);
    form_name = (text*)strtok(buf, ".");
    /* Initialize the attribute mask */
    d2fctx_attr.mask_d2fctxa = 0;
    /* for MS Windows-only attributes */
    d2fctx_attr.d2fihnd_d2fctxa = hInstance;
    d2fctx_attr.d2fphnd_d2fctxa = hPrevInstance;
    d2fctx_attr.d2fcmsh_d2fctxa = cmdShow;
    /* Create the API context */
    status = d2fctxcr_Create(&pd2fctx, &d2fctx_attr);
    /* Create the context */
    d2fctxcn_Connect(pd2fctx, (text*)"scott/tiger@test");
    /* Create the form */
    d2ffmdcr_Create(pd2fctx, pd2ffmd, form_name);
    /* Create a window */
    d2fwinrcr_Create(pd2fctx, pd2ffmd, pd2fwin, (text*)"MYWIN");
    /* Create Canvas and set canvas-related properties */
    /* Create a canvas */
```

6-298  Developer/2000: Guidelines for Building Applications
6.3 Using the Open API to Build and Modify Form Builder Applications

```c

d2fcnvcr_Create(pd2fctxt, pd2ffmd, &pd2fcnv, (text*)"MYCANVAS");
/* Set viewport width */
d2fcnvs_vprt_wid(pd2fctxt, pd2fcnv, 512);
/* Set viewport height */
d2fcnvs_vprt_hgt(pd2fctxt, pd2fcnv, 403);
/* Set window */
d2fcnvs wnd_obj(pd2fctxt, pd2fcnv, pd2fwin);
/* Set viewport X-position */
d2fcnvs_vprt_x_pos(pd2fctxt, pd2fcnv, 0);
/* Set viewport Y-position */
d2fcnvs_vprt_y_pos(pd2fctxt, pd2fcnv, 0);
/* Set width */
d2fcnvs_width(pd2fctxt, pd2fcnv, 538)
/* Set height */
d2fcnvs_height(pd2fctxt, pd2fcnv, 403)
/*** Create Emp block and set block-related properties ***/
/* Create block */
d2fbblkcr_Create(pd2fctxt, pd2ffmd, &pempblk, (text*)"EMP");
/* Set to database block */
d2fbblks_db_blk(pd2fctxt, pemppblk, TRUE);
/* Set query data source to Table */
d2fbblks_qry_dat_src_typ(pd2fctxt, pemppblk, D2FC_QRDA_TABLE);
/* Set query data source name to EMP table */
d2fbblks_qry_dat_src_nam(pd2fctxt, pemppblk, "EMP");
/* Set DML data source type to Table */
d2fbblks_dml_dat_typ(pd2fctxt, pemppblk, D2FC_DMDA_TABLE);
/* Set DML data source name to EMP table */
d2fbblks_dml_dat_nam(pd2fctxt, pemppblk, (text*)"EMP");
/*** Create Dept block and set block-related properties ***/
/* Create block */
d2fbblkcr_Create(pd2fctxt, pd2ffmd, &pdeptblk, (text*)"DEPT");
/* Set to database block */
d2fbblks_db_blk(pd2fctxt, pdeptblk, TRUE);
/* Set query data source to Table */
d2fbblks_qry_dat_src_typ(pd2fctxt, pdeptblk, D2FC_QRDA_TABLE);
/* Set query data source name to EMP table */
d2fbblks_qry_dat_src_nam(pd2fctxt, pdeptblk, "DEPT");
/* Set DML data source type to Table */
d2fbblks_dml_dat_typ(pd2fctxt, pdeptblk, D2FC_DMDA_TABLE);
/* Set DML data source name to EMP table */
d2fbblks_dml_dat_nam(pd2fctxt, pdeptblk, (text*)"DEPT");
/*** Create empno item and item-related properties ***/
/* Create item */
d2fitmcr_Create(pd2fctxt, pemppblk, &pEempnoitm, (text*)"EMPNO");
/* Set item type */
```
d2fitms_itm_type(pd2fctx, pEempnoitm, D2FC_ITTY_TI);
/* Set Enable property */
d2fitms_enabled(pd2fctx, pEempnoitm, TRUE);
/* Set item (keyboard) navigable property */
d2fitms_kbd_navigable(pd2fctx, pEempnoitm, TRUE);
/* Set item Data Type property */
d2fitms_dat_typ(pd2fctx, pEempnoitm, D2FC_DATY_NUMBER);
/* Set item Max Length property */
d2fitms_max_len(pd2fctx, pEempnoitm, 6);
/* Set item Required property */
d2fitms_required(pd2fctx, pEempnoitm, TRUE);
/* Set Distance Between Records property */
d2fitms_dist_btwn_recs(pd2fctx, pEempnoitm, 0);
/* Set Database block(Database Item) property */
d2fitms_db_itm(pd2fctx, pEempnoitm, TRUE);
/* Set Query Allowed */
d2fitms_qry_allowed(pd2fctx, pEempnoitm, TRUE);
/* Set Query Length */
d2fitms_qry_len(pd2fctx, pEempnoitm, 6);
/* Set Update Allowed */
d2fitms_updt_allowed(pd2fctx, pEempnoitm, TRUE);
/* Set Item Displayed (Visible) */
d2fitms_visible(pd2fctx, pEempnoitm, TRUE);
/* Set Item Canvas property */
d2fitms_cnv_obj(pd2fctx, pEempnoitm, pd2fcnv);
/* Set Item X-position */
d2fitms_x_pos(pd2fctx, pEempnoitm, 32);
/* Set Item Y-position */
d2fitms_y_pos(pd2fctx, pEempnoitm, 50);
/* Set Item Width */
d2fitms_width(pd2fctx, pEempnoitm, 51);
/* Set Item Height */
d2fitms_height(pd2fctx, pEempnoitm, 17);
/* Set Item Bevel */
d2fitms_bevel(pd2fctx, pEempnoitm, D2FC_BEST_LOWERED);
/* Set item Hint */
d2fitms_hint(pd2fctx, pEempnoitm, (text*)"Enter value for :EMPNO");
/*** Create Ename item and item-related properties ***/
/* Create item */
d2fitmc_r_create(pd2fctx, pempblk, &pEenameitm, (text*)"ENAME");
/* Set item type */
d2fitms_itm_type(pd2fctx, pEenameitm, D2FC_ITTY_TI);
/* Set Enable property */
d2fitms_enabled(pd2fctx, pEenameitm, TRUE);
/* Set item (keyboard) navigable property */
6.3 Using the Open API to Build and Modify Form Builder Applications

d2fitms_kbrd_navigable(pd2fctx, pEenameitm, TRUE);
/* Set item Data Type property */
d2fitms_dat_typ(pd2fctx, pEenameitm, D2FC_DATY_CHAR);
/* Set item Max Length property */
d2fitms_max_len(pd2fctx, pEenameitm, 10);
/* Set Distance Between Records property */
d2fitms_dist BTN_recs(pd2fctx, pEenameitm, 0);
/* Set Database block (Database Item) property */
d2fitms_db itm(pd2fctx, pEenameitm, TRUE);
/* Set Query Allowed */
d2fitms_qry_allowed(pd2fctx, pEenameitm, TRUE);
/* Set Query Length */
d2fitms_qry_len(pd2fctx, pEenameitm, 10);
/* Set Update Allowed */
d2fitms_updt_allowed(pd2fctx, pEenameitm, TRUE);
/* Set Item Displayed (Visible) */
d2fitms_visible(pd2fctx, pEenameitm, TRUE);
/* Set Item Canvas property */
d2fitms_cnv_obj(pd2fctx, pEenameitm, pd2fcnv);
/* Set Item X-position */
d2fitms_x_pos(pd2fctx, pEenameitm, 83);
/* Set Item Y-position */
d2fitms_y_pos(pd2fctx, pEenameitm, 50);
/* Set Item Width */
d2fitms_width(pd2fctx, pEenameitm, 77);
/* Set Item Height */
d2fitms_height(pd2fctx, pEenameitm, 17);
/* Set Item Bevel */
d2fitms_bevel(pd2fctx, pEenameitm, D2FC_BESTLOWERED);
/* Set Item Hint */
d2fitms_hint(pd2fctx, pEenameitm, (text*)"Enter value for :ENAME");
/*** Create JOB item and item-related properties ***/
/* Create item */
d2fitmcr_Create(pd2fctx, pempblk, &pEjobitm, (text*)"JOB");
/* Set item type */
d2fitms_itm_type(pd2fctx, pEjobitm, D2FC_ITTY_TI);
/* Set Enable property */
d2fitms_enabled(pd2fctx, pEjobitm, TRUE);
/* Set item (keyboard) navigable property */
d2fitms_kbrd_navigable(pd2fctx, pEjobitm, TRUE);
/* Set item Data Type property */
d2fitms_dat_typ(pd2fctx, pEjobitm, TRUE);
/* Set item Max Length property */
d2fitms_max_len(pd2fctx, pEjobitm, 9);
/* Set Distance Between Records property */
6.3 Using the Open API to Build and Modify Form Builder Applications

d2fitms_dist_btwn_recs(pd2fctx, pEjobitm, 0);
/* Set Database block(Database Item) property */
d2fitms_db_itm(pd2fctx, pEjobitm, TRUE);
/* Set Query Allowed */
d2fitms_qry_allowed(pd2fctx, pEjobitm, TRUE);
/* Set Query Length */
d2fitms_qry_len(pd2fctx, pEjobitm, 9);
/* Set Update Allowed */
d2fitms_updt_allowed(pd2fctx, pEjobitm, TRUE);
/* Set Item Displayed (Visible) */
d2fitms_visible(pd2fctx, pEjobitm, TRUE);
/* Set Item Canvas property */
d2fitms_cnv_obj(pd2fctx, pEjobitm, pd2fcnv);
/* Set Item X-position */
d2fitms_x_pos(pd2fctx, pEjobitm, 160);
/* Set Item Y-position */
d2fitms_y_pos(pd2fctx, pEjobitm, 50);
/* Set Item Width */
d2fitms_width(pd2fctx, pEjobitm, 70);
/* Set Item Height */
d2fitms_height(pd2fctx, pEjobitm, 17);
/* Set Item Bevel */
d2fitms_bevel(pd2fctx, pEjobitm, D2FC_BEST_LOWERED);
/* Set item Hint */
d2fitms_hint(pd2fctx, pEjobitm, (text*)"Enter value for :JOB");
/*** Create SALARY item and item-related properties ***/
/* Create item */
d2fitmcr_Create(pd2fctx, pemblk, &pEsalitm, (text*)"SAL");
/* Set item type */
d2fitms_itm_type(pd2fctx, pEsalitm, D2FC_ITTY_TI);
/* Set Enable property */
d2fitms_enabled(pd2fctx, pEsalitm, TRUE);
/* Set item (keyboard) navigable property */
d2fitms_kbrd_navigable(pd2fctx, pEsalitm, TRUE);
/* Set item Data Type property */
d2fitms_dat_typ(pd2fctx, pEsalitm, D2FC_DATY_NUMBER);
/* Set item Max Length property */
d2fitms_max_len(pd2fctx, pEsalitm, 9);
/* Set Distance Between Records property */
d2fitms_dist_btwn_recs(pd2fctx, pEsalitm, 0);
/* Set Database block(Database Item) property */
d2fitms_db_itm(pd2fctx, pEsalitm, TRUE);
/* Set Query Allowed */
d2fitms_qry_allowed(pd2fctx, pEsalitm, TRUE);
/* Set Query Length */
6.3 Using the Open API to Build and Modify Form Builder Applications

```c
    d2fitms_qry_len(pd2fctxt, pEsalitm, 9);
    /* Set Update Allowed */
    d2fitms_updt_allowed(pd2fctxt, pEsalitm, TRUE);
    /* Set Item Displayed (Visible) */
    d2fitms_visible(pd2fctxt, pEsalitm, TRUE);
    /* Set Item Canvas property */
    d2fitms_cvn_obj(pd2fctxt, pEsalitm, pd2fcnv);
    /* Set Item X-position */
    d2fitms_x_pos(pd2fctxt, pEsalitm, 352);
    /* Set Item Y-position */
    d2fitms_y_pos(pd2fctxt, pEsalitm, 50);
    /* Set Item Width */
    d2fitms_width(pd2fctxt, pEsalitm, 70);
    /* Set Item Height */
    d2fitms_height(pd2fctxt, pEsalitm, 17);
    /* Set Item Bevel */
    d2fitms_bevel(pd2fctxt, pEsalitm, D2FC_BEST_LOWERED);
    /* Set item Hint */
    d2fitms_hint(pd2fctxt, pEsalitm, (text*)"Enter value for :SAL");
    /** Create DEPTNO item and item-related properties **/
    /* Create item */
    d2fitmcr_Create(pd2fctxt, pempblk, &pEdeptnoitm, (text*)"DEPTNO");
    /* Set item type */
    d2fitms_itm_type(pd2fctxt, pEdeptnoitm, D2FC_ITTY_TI);
    /* Set Enable property */
    d2fitms_enabled(pd2fctxt, pEdeptnoitm, TRUE);
    /* Set item (keyboard) navigable property */
    d2fitms_kbrd_navigable(pd2fctxt, pEdeptnoitm, TRUE);
    /* Set item Data Type property */
    d2fitms_dat_typ(pd2fctxt, pEdeptnoitm, D2FC_DATY_NUMBER);
    /* Set item Max Length property */
    d2fitms_max_len(pd2fctxt, pEdeptnoitm, 4);
    /* Set item Required property */
    d2fitms_required(pd2fctxt, pEdeptnoitm, TRUE);
    /* Set Distance Between Records property */
    d2fitms_dist btwn_recs(pd2fctxt, pEdeptnoitm, 0);
    /* Set Database block (Database Item) property */
    d2fitms_db_item(pd2fctxt, pEdeptnoitm, TRUE);
    /* Set Query Allowed */
    d2fitms_qry_allowed(pd2fctxt, pEdeptnoitm, TRUE);
    /* Set Query Length */
    d2fitms_qry_len(pd2fctxt, pEdeptnoitm, 4);
    /* Set Update Allowed */
    d2fitms_updt_allowed(pd2fctxt, pEdeptnoitm, TRUE);
    /* Set Item Displayed (Visible) */
```
6.3 Using the Open API to Build and Modify Form Builder Applications

d2fitms_visible(pd2fctx, pEdeptnoitm, TRUE);
/* Set Item Canvas property */
d2fitms_cnv_obj(pd2fctx, pEdeptnoitm, pd2fcnv);
/* Set Item X-position */
d2fitms_x_pos(pd2fctx, pEdeptnoitm, 493);
/* Set Item Y-position */
d2fitms_y_pos(pd2fctx, pEdeptnoitm, 50);
/* Set Item Width */
d2fitms_width(pd2fctx, pEdeptnoitm, 30);
/* Set Item Height */
d2fitms_height(pd2fctx, pEdeptnoitm, 17);
/* Set Item Bevel */
d2fitms_bevel(pd2fctx, pEdeptnoitm, D2FC_BEST_LOWERED);
/* Set item Hint */
d2fitms_hint(pd2fctx, pEdeptnoitm, (text*)"Enter value for :DEPTNO");
/*** Create DEPTNO item and item-related properties ***/
/* Create item */
d2fitmcr_Create(pd2fctx, pdeptblk, &pDdeptnoitm, (text*)"DEPTNO");
/* Set item type */
d2fitms_itm_type(pd2fctx, pDdeptnoitm, D2FC_ITTY_TI);
/* Set Enable property */
d2fitms_enabled(pd2fctx, pDdeptnoitm, TRUE);
/* Set item (keyboard) navigable property */
d2fitms_kbrd_navigable(pd2fctx, pDdeptnoitm, TRUE);
/* Set item Data Type property */
d2fitms_dat_typ(pd2fctx, pDdeptnoitm, D2FC_DATY_NUMBER);
/* Set item Max Length property */
d2fitms_max_len(pd2fctx, pDdeptnoitm, 4);
/* Set item Required property */
d2fitms_required(pd2fctx, pDdeptnoitm, TRUE);
/* Set Distance Between Records property */
d2fitms_dist_btwn_recs(pd2fctx, pDdeptnoitm, 0);
/* Set Database block (Database Item) property */
d2fitms_db_itm(pd2fctx, pDdeptnoitm, TRUE);
/* Set Query Allowed */
d2fitms_qry_allowed(pd2fctx, pDdeptnoitm, TRUE);
/* Set Query Length */
d2fitms_qry_len(pd2fctx, pDdeptnoitm, 4);
/* Set Update Allowed */
d2fitms_updtt_allowed(pd2fctx, pDdeptnoitm, TRUE);
/* Set Item Displayed (Visible) */
d2fitms_visible(pd2fctx, pDdeptnoitm, TRUE);
/* Set Item Canvas property */
d2fitms_cnv_obj(pd2fctx, pDdeptnoitm, pd2fcnv);
/* Set Item X-position */
Taking Advantage of Developer/2000’s Open Architecture

6.3 Using the Open API to Build and Modify Form Builder Applications

d2fitms_x_pos(pd2fctx, pDdeptnoitm, 32);
/* Set Item Y-position */
d2fitms_y_pos(pd2fctx, pDdeptnoitm, 151);
/* Set Item Width */
d2fitms_width(pd2fctx, pDdeptnoitm, 38);
/* Set Item Height */
d2fitms_height(pd2fctx, pDdeptnoitm, 17);
/* Set Item Bevel */
d2fitms_bevel(pd2fctx, pDdeptnoitm, D2FC_BEST_LOWERED);
/* Set item Hint */
d2fitms_hint(pd2fctx, pDdeptnoitm, (text*)"Enter value for :DEPTNO");
/*** Create DNAME item and item-related properties ***/
/* Create item */
d2fitmcr_Create(pd2fctx, pdeptblk, &pDdnameitm, (text*)"DNAME");
/* Set item type */
d2fitsms_itm_type(pd2fctx, pDdnameitm, D2FC_ITTY_TI);
/* Set Enable property */
d2fitms_enabled(pd2fctx, pDdnameitm, TRUE);
/* Set item (keyboard) navigable property */
d2fitsms_kbrd_navigable(pd2fctx, pDdnameitm, TRUE);
/* Set item Data Type property */
d2fitsms_dat_typ(pd2fctx, pDdnameitm, D2FC_DATY_CHAR);
/* Set item Max Length property */
d2fitms_max_len(pd2fctx, pDdnameitm, 14);
/* Set Distance Between Records property */
d2fitms_dist_btwn_recs(pd2fctx, pDdnameitm, 0);
/* Set Database block(Database Item) property */
d2fitms_db_itm(pd2fctx, pDdnameitm, TRUE);
/* Set Query Allowed */
d2fitsms_qry_allowed(pd2fctx, pDdnameitm, TRUE);
/* Set Query Length */
d2fitsms_qry_len(pd2fctx, pDdnameitm, 14);
/* Set Update Allowed */
d2fitsms_updt_allowed(pd2fctx, pDdnameitm, TRUE);
/* Set Item Displayed (Visible) */
d2fitsms_visible(pd2fctx, pDdnameitm, TRUE);
/* Set Item Canvas property */
d2fitsms_cnv_obj(pd2fctx, pDdnameitm, pd2fcnv);
/* Set Item X-position */
d2fitsms_x_pos(pd2fctx, pDdnameitm, 70);
/* Set Item Y-position */
d2fitsms_y_pos(pd2fctx, pDdnameitm, 151);
/* Set Item Width */
d2fitsms_width(pd2fctx, pDdnameitm, 102);
/* Set Item Height */
d2fitms_height(pd2fctx, pdnameitm, 17);
/* Set Item Bevel */
d2fitms_bevel(pd2fctx, pdnameitm, D2FC_BEST_LOWERED);
/* Set item Hint */
d2fitms_hint(pd2fctx, pdnameitm, (text*)"Enter value for :DNAME");
/*** Create LOC item and item-related properties ***/
/* Create item */
d2fitmc_Create(pd2fctx, pdeptblk, &plocitm, (text*)"LOC");
/* Set item type */
d2fitms_itm_type(pd2fctx, plocitm, D2FC_ITTY_TI);
/* Set Enable property */
d2fitms_enabled(pd2fctx, plocitm, TRUE);
/* Set item (keyboard) navigable property */
d2fitms_kbd_navigable(pd2fctx, plocitm, TRUE);
/* Set item Data Type property */
d2fitms_dat_typ(pd2fctx, plocitm, D2FC_DATY_CHAR);
/* Set item Max Length property */
d2fitms_max_len(pd2fctx, plocitm, 13);
/* Set Distance Between Records property */
d2fitms_dist_btwn_recs(pd2fctx, plocitm, 0);
/* Set Database block (Database Item) property */
d2fitms_db_itm(pd2fctx, plocitm, TRUE);
/* Set Query Allowed */
d2fitms_qry_allowed(pd2fctx, plocitm, TRUE);
/* Set Query Length */
d2fitms_qry_len(pd2fctx, plocitm, 13);
/* Set Update Allowed */
d2fitms_updlt_allowed(pd2fctx, plocitm, TRUE);
/* Set Item Displayed (Visible) */
d2fitms_visible(pd2fctx, plocitm, TRUE);
/* Set Item Canvas property */
d2fitms_cnv_obj(pd2fctx, plocitm, pd2fcnv);
/* Set Item X-position */
d2fitms_x_pos(pd2fctx, plocitm, 173);
/* Set Item Y-position */
d2fitms_y_pos(pd2fctx, plocitm, 151);
/* Set Item Width */
d2fitms_width(pd2fctx, plocitm, 96);
/* Set Item Height */
d2fitms_height(pd2fctx, plocitm, 17);
/* Set Item Bevel */
d2fitms_bevel(pd2fctx, plocitm, D2FC_BEST_LOWERED);
/* Set item Hint */
d2fitms_hint(pd2fctx, plocitm, (text*)"Enter value for :LOC");
/*** Create Relations and relations-related properties ***/
/* Create Relation */
d2frelcr_Create(pd2fctx, (d2fob *)pdeptblk, &pd2frel, (text*)"DEPT_EMP");
/* Set Relation Detail block */
d2frels_detail_blk(pd2fctx, pd2frel, (text*)"EMP");
/* Set Master Deletes property */
d2frels_del_rec(pd2fctx, pd2frel, D2FC_DEREC_NON_ISOLATED);
/* Set Deferred property */
d2frels_deferred(pd2fctx, pd2frel, FALSE);
/* Set Auto Query property */
d2frels_auto_qry(pd2fctx, pd2frel, FALSE);
/* Set Prevent Masterless property */
d2frels_prvnt_mstrless_ops(pd2fctx, pd2frel, FALSE);
/* Set Join Condition property */
d2frels_join_cond(pd2fctx, pd2frel, (text*)"DEPTNO");
/* Instantiate Relation: creates master-detail triggers */
d2frelup_Update(pd2fctx, pd2frel);
/* Save Form */
d2ffmdsv_Save(pd2fctx, pd2ffmd, (text*)0, FALSE, TRUE);
/* Compile Form */
d2ffmdcf_CompileFile(pd2fctx, pd2ffmd);
/* Destroy Context */
d2fctxde_Destroy(pd2fctx);
6.4 Designing Applications to Run against ODBC Datasources

The data within your enterprise often resides within several heterogeneous data-sources. Some portion of your data, for example, might be stored within an Oracle database, while another portion is stored within an Informix database. Building a single application that can access each datasource can be a difficult task.

However, by taking advantage of Developer/2000’s open datasource support, you can build generic applications that run transparently against any ODBC-compliant datasource.

This section describes Developer/2000’s open datasource support. It includes these topics:

- Section 6.4.1, “What is the Oracle Open Client Adapter (OCA)?”
- Section 6.4.2, “Open datasource guidelines”
- Section 6.4.3, “Configuring Developer/2000 to run against ODBC datasources”

6.4.1 What is the Oracle Open Client Adapter (OCA)?

When you connect to an ODBC datasource, you use the Oracle Open Client Adapter (OCA). OCA is an ODBC level 2-compliant utility that allows Developer/2000 on Microsoft Windows 95, Windows NT, and Windows 3.1 to access ODBC-compliant datasources through ODBC drivers.

OCA is included with Developer/2000. You use the Oracle Installer to install OCA.

6.4.1.1 When should I use OCA?

You should use OCA whenever your application must access non-Oracle data-sources. Developer/2000 applications can automatically access any ODBC-compliant datasources. Refer to the online help for specific information about connecting to ODBC datasources.
6.4.1.2 OCA architecture
The Oracle Open Client Adapter consists of the following:

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Developer/2000 Application</td>
<td>Performs processing and calls ODBC functions to submit SQL statements and retrieve results.</td>
</tr>
<tr>
<td>Oracle Open Client Adapter</td>
<td>Translates Oracle database calls to ODBC calls.</td>
</tr>
<tr>
<td>Driver Manager</td>
<td>Loads ODBC drivers for your application.</td>
</tr>
<tr>
<td>ODBC Drivers</td>
<td>Process ODBC function calls, submits SQL requests to a specific datasource, and returns results to your application.</td>
</tr>
<tr>
<td>Datasource</td>
<td>Consists of the data that user wants to access and its associated operating system, DBMS, and network platform (if any) used to access the DBMS.</td>
</tr>
</tbody>
</table>

6.4.1.3 Establishing an ODBC connection
To connect to an ODBC datasource, type the following connect string in the Connect dialog box:

```
[user[/password]]@ODBC:datasource[:dbname]
```

For example, to connect to Sybase System 10, type:

```
scott/tiger@ODBC:sybase_ds
```

6.4.1.4 ODBC drivers
When you connect to an ODBC datasource, you use an ODBC driver to communicate with the datasource. Developer/2000 includes prebundled ODBC drivers for each supported datasource. These drivers are ODBC level 1-compliant and, to some extent, provide some level 2 functionality to achieve greater performance.
6.4 Designing Applications to Run against ODBC Datasources

6.4.1.5 OPENDB.PLL

OPENDB.PLL is a PL/SQL library of functions that is included with OCA. You use OPENDB.PLL within Developer/2000 applications to:

- Automatically adjust form and data block properties at runtime to suit the datasource.
- Open auxiliary connections to other datasources in addition to the application's main connection.
- Execute arbitrary SQL statements and Stored Procedure calls on any connection.
- Retrieve results sets from non-Oracle stored procedures.
- Obtain the DBMS and ODBC driver names and versions for a given connection.

For more information about OPENDB.PLL, refer to OCA_INFO.PDF in the ORACLE_HOME\TOOLS\DOC20 directory.

6.4.2 Open datasource guidelines

When working with multiple datasources, consider these guidelines:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Recommendation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimizing your application to run against multi-</td>
<td>You do not have to optimize your application to run against multiple datasources unless you want to target a specific datasource to take advantage of features particular to that system.</td>
</tr>
<tr>
<td>ple datasources</td>
<td></td>
</tr>
<tr>
<td>Writing PL/SQL for use with ODBC datasources</td>
<td>SQL statements embedded in PL/SQL program units must conform to both Oracle SQL and the SQL dialect of the datasource that you connect against. Any statements that fail against Oracle will cause PL/SQL compilation failures. Similarly, any statements that use unsupported syntax will fail at execution. The SYSDATE and USER functions are the only exceptions to this restriction. These functions are Oracle-specific; and, OCA translates these functions to the corresponding ODBC functions, allowing these functions to work against all datasources. If you want to issue SQL statements that are datasource-specific, but conflict with Oracle syntax, use the EXEC_SQL package.</td>
</tr>
</tbody>
</table>
Referencing tables from more than one datasource

Many datasources allow you to access tables that are located in other datasources if you specify the database, owner, and table (for example, database.owner.tablename).

PL/SQL does not recognize the three-part table syntax, and your client-side program unit or trigger will not compile.

To work around this restriction, enclose the three-part name in double quotes, after calling the appropriate OPENDB function that removes double quotes.

Restrictions

- When working with a non-Oracle7 datasource, you must store your Developer/2000 modules (forms, reports, and graphics) in the file system. Non-Oracle7 datasources cannot be used as a repository for storing Developer/2000 modules.

- Trigger information for columns cannot be accessed from the Object Navigator (Database Objects node).

- You can view stored procedure text only for datasources that emulate the Oracle ALL_SOURCE table (e.g., Microsoft SQL Server). You cannot edit database stored procedure text.

- You cannot drag and drop PL/SQL program units from the client to a non-Oracle7 datasource.

- Developer/2000 cannot use primary and foreign key constraint information of OCA datasources for default selection of master-detail relationships. These relationships must be identified directly where required.

- Optimizer hints (*hint*/ style comments) are ignored by any datasource that you connect to through OCA.

Troubleshooting

If an application that used to work against Oracle fails against an ODBC datasource, the most likely explanation is that you have encountered a restriction or limitation of either ODBC or the datasource when used with Developer/2000.

To view the SQL statements issued by OCA and the messages generated by the ODBC driver or the database:

1. Verify that the following entry is set in the ORACLE.INI file on Windows 3.1 or in the registry on Windows NT and Windows 95:
   
   `UB=ORACLE_HOME\OCA20`
6.4 Designing Applications to Run against ODBC Datasources

<table>
<thead>
<tr>
<th>Topic</th>
<th>Recommendation</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.</td>
<td>If you are unable to resolve the error, call Oracle Customer Support.</td>
</tr>
</tbody>
</table>
| 3.    | Add the following entries to the ORACLE.INI file on Windows 3.1 or to the registry under SOFTWARE\ORACLE on Windows NT and Windows 95:  
      OCA_DEBUG_SQL=TRUE  
      OCA_DEBUG_ERROR=TRUE |
| 4.    | Run your application against the ODBC datasource to view SQL statements or error messages in the debug window. Click OK to close the debug window and continue processing. |

**Debugging Tips**  
You can display debug information by setting the OCA_DEBUG_SQL and OCA_DEBUG_ERROR environment variables to TRUE.  
Using these environment variables will help you identify SQL failures when using Developer/2000 against OCA datasources.  
When you set OCA_DEBUG to TRUE, any SQL statement that is sent to the ODBC driver is displayed before it is transmitted.  
When you set OCA_DEBUG_ERROR to TRUE, any errors that are returned by the ODBC driver are displayed in a dialog before being passed back to Developer/2000.  

6.4.3 Configuring Developer/2000 to Run against an ODBC Datasource

To configure Developer/2000 to run against an ODBC-compliant datasource, refer to the “Accessing non-Oracle datasources” topic in the online help.
Part 2
Appendices
This appendix covers these topics:

- Section A.1, “About Template HTML Files”
- Section A.2, “Working with the Cartridge HTML File Template”
- Section A.3, “Working with the Static (Non-Cartridge) HTML File Template”

A.1 About Template HTML Files

The Developer/2000 Server includes two “template” HTML files for your use in deploying Form Builder applications on the Web:

- CARTRIDGE.HTML (cartridge HTML file template)
- STATIC.HTML (static, non-cartridge HTML file template)

Any time you wish to base an HTML file on a template, simply copy the appropriate template to a new filename, then customize the new file to fit your application(s).

Note: Due to an HTML syntax requirement, you must locate the tags that define the Forms Client applet between the <BODY> and </BODY> tags of an HTML file.
A.2 Working with the Cartridge HTML File Template

When you deploy a Form Builder application on the Web in a cartridge implementation, you can create a cartridge HTML file for the application. The easiest way to do this is to modify the cartridge HTML file template:

ORACLE_HOME\tools\demo20\cartridg.html

At runtime, the Forms Cartridge Handler dynamically creates a new HTML file by merging information from the following sources:

- the application’s cartridge HTML file
- the application’s cartridge settings
- the application’s URL

The new (dynamically-created) HTML file then downloads to the end user’s Web browser.

For more information, refer to the following sections:

- Section 2.2.1.3.4, “Creating an application cartridge”
- Section 2.2.1.3.6, “Step 3c: Providing URLs of your Web-enabled applications”
A.2 Working with the Cartridge HTML File Template

A.2.1 Snapshot: the cartridge HTML file template

Note: Carriage returns have been added to this example to make it more readable. When working with template files, do not add carriage returns. Refer to the online version of this file for more information.

```html
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>The Developer/2000 Server</title>
  </head>
  <body>
    Please wait while the Forms Client class files download and run. This will take a second or two...
  </body>
</html>
```

<!--- file: cartridge.html -->
<!--- Oracle Cartridge HTML File Template -->
<!--- Rename, and modify tags and parameter values as needed -->

<!head>
<title>The Developer/2000 Server</title>
</head>

<body>
Please wait while the Forms Client class files download and run. This will take a second or two...
</body>

<!-- applet definition (start) -->
<applet codebase="%LEASTLOADEDHOST%/codebase_vdir/"
  code="oracle.forms.uiClient.v1_4.engine.Main"
  archive="%LEASTLOADEDHOST%/jars_vdir/f50web.jar"
  height=20
  width=20>
  <param name="serverPort" value="9000">
  <param name="serverArgs"
    value="forms_param=%forms_param%
      user_param=%user_param%">
  <param name="serverApp" value="default">
</applet>
<!-- applet definition (end) -->

</body>
</html>
A.2.2 Modifying the cartridge HTML file template

Modify the Oracle cartridge HTML file template to fit your application as follows:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Parameter Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>codebase</td>
<td>required Replace /codebase_vdir/ with the virtual directory you defined to point to the physical directory ORACLE_HOME/forms50/java. For example: /web_code/</td>
</tr>
<tr>
<td>archive</td>
<td>optional Replace or add to /jars_vdir/f50web.jar to provide the virtual directory path and filename of any JAR file(s) you want to be downloaded to end users’ Web browsers at application startup. For more information, refer to Section 2.2.1.4, “Configuring the Forms Client”.</td>
</tr>
<tr>
<td>serverPort</td>
<td>optional Replace 9000 with the number of the port on which the Forms Server Listener process was started. In most cases, the port number will remain 9000 (the default). For more information, refer to Section 2.2.1.2, “Starting and stopping the Forms Listener Server”.</td>
</tr>
<tr>
<td>serverArgs</td>
<td>optional Replace forms_param with any valid Form Builder command-line parameter. Replace user_param with any valid user-defined parameter.</td>
</tr>
<tr>
<td>serverApp</td>
<td>optional Replace default with the name of your application class (if any). Use application classes for creating application-specific font mapping and icon path settings. For more information, refer to Section B.2, “Referencing an application class”</td>
</tr>
<tr>
<td>leastloadedhost</td>
<td>optional During load balancing, this fixed-name place holder is replaced dynamically by the cartridge with the name of the least-loaded system. The cartridge gets this information from the Metrics Server at runtime.</td>
</tr>
</tbody>
</table>

Table 6–1 Cartridge HTML file parameters and values
You can provide a delimited value for any cartridge HTML file parameter. For example, you could have the following line in your cartridge HTML file:

```
ARCHIVE="%Archive%"
```

You then must assign a value to `%Archive%` (either in the application’s cartridge settings, or in the application’s URL).

**Note:** All variable parameters in your cartridge HTML file must receive values in this way at runtime.

If a parameter does not receive a value, the Forms Cartridge Handler cannot build an HTML file to pass back to the user’s Web browser, resulting in an error. For information on checking for Forms Cartridge Handler errors, refer to Section 2.2.3, “Troubleshooting your Form Builder configuration”.

**Note:** If you are using load balancing and you use the ARCHIVE parameter to point to a physical directory where Oracle JAR files (and any custom JAR files) are stored, this URL must be the same as the URL for CODEBASE since everything must be downloaded from the same system. Thus, if you use `%LEASTLOADEDHOST%` in your CODEBASE definition, use it in your ARCHIVE definition as well. For example, if your old HTML file had the following entry for ARCHIVE:

```
ARCHIVE="jars_vdir/f45web.jar"
```

Then your new HTML file will have the following entry for ARCHIVE:

```
ARCHIVE="%LEASTLOADEDHOST%/jars_vdir/f45web.jar"
```

### A.2.3 Examples of customized cartridge HTML files

The following examples display variations of the cartridge HTML file template.

#### A.2.3.1 Example 1

ACME has created a cartridge HTML file template. It includes the following:

- a title for the Web browser window
- a virtual directory for the Form Builder Java codebase: `/web_code/`
- a Form Builder command-line parameter: Module
Note: Carriage returns have been added to this example to make it more readable. When working with template files, do not add carriage returns. Refer to the online version of this file for more information.

```html
<html>
<!-- FILE: tempcart.html -->
<!-- ACME cartridge HTML file template -->
<!-- rename and modify as needed -->

<head>
<title>ACME Web Applications</title>
</head>

<body>
<center>
<br>Welcome to ACME Web Applications!
<br>Please wait for the application to load...
<br></center>

<applet codebase="/web_code/
    code="oracle.forms.uiClient.v1_4.engine.Main"
    height=20 width=20>
    <param name="serverPort" value="9000">
    <param name="serverArgs" value="Module=%Module%">
</applet>

</body>
</html>

A.2.3.2 Example 2
General Traders Inc.'s cartridge HTML file includes the following:

- a virtual directory for images: imag
- a background image and main image: gti_wall.jpg and gti_logo.jpg
- a virtual directory for JAR files: /web_jars/
- two JAR files: f50web.jar and gti.jar
- a virtual directory for the Form Builder Java codebase: /web_code/
- two Form Builder command-line parameters: Module and Userid
- one user-defined parameter: Dept_id
A.3  Working with the Static (Non-Cartridge) HTML File Template

- an application class setting: `gti_ac`
- the parameter to specify that load balancing be used: `%leastloadedhost%

**Note:** Carriage returns have been added to this example to make it more readable. When working with template files, do not add carriage returns. Refer to the online version of this file for more information.

```html
<html>
<!-- FILE: gti_cart.html -->
<!-- General Traders cartridge HTML file -->

<body background="/imag/gti_wall.jpg">
<center><img src="/imag/gti_logo.gif" height=170 width=120 alt="gti_logo.jpg"></center>
<p>
<applet codebase="%leastloadedhost%/web_code/"
   code="oracle.forms.uiClient.v1_4.engine.Main"
   archive="%leastloadedhost%/web_jars/f50web.jar , /web_jars/gti.jar"
   height=20 width=20>
   <param name="serverPort" value="7777">
   <param name="serverArgs" value="Module=%Module% Userid=%Userid% Dept_id=%Dept_id%">
   <param name="serverApp" value="gti_ac">
</applet>
</body>
</html>
```

**A.3 Working with the Static (Non-Cartridge) HTML File Template**

When you deploy a Form Builder application on the Web in a non-cartridge implementation, you must create a static HTML file for the application. The easiest way to do this is to modify the Oracle static HTML file template:

`ORACLE_HOME/forms50/demos/webdemos/static.html`

At runtime, the static HTML file downloads to the end user’s Web browser.
For more information, refer to the following sections:

- Section 2.2.1.3.5, “Creating a static (non-cartridge) HTML file”
- Section 2.2.1.3.6, “Step 3c: Providing URLs of your Web-enabled applications”

A.3.1 Snapshot: the static (non-cartridge) HTML file template

Note: Carriage returns have been added to this example to make it more readable. When working with template files, do not add carriage returns. Refer to the online version of this file for more information.

```html
<HTML>
<!-- FILE: static.html -->
<!-- Oracle Static (Non-Cartridge) HTML File Template -->
<!-- Rename, and modify tags and parameter values as needed -->

<HEAD><TITLE>The Developer/2000 Server</TITLE></HEAD>

<BODY>
Please wait while the Forms Client class files download and run.
This will take a second or two...

<!-- applet definition (start) -->
<APPLET CODEBASE="/codebase_vdir/
CODE="oracle.forms.uiClient.v1_4.engine.Main"
ARCHIVE="/jars_vdir/f50web.jar"
HEIGHT=20
WIDTH=20>
<PARAM NAME="serverPort" VALUE="9000">
<PARAM NAME="serverArgs"
VALUE="module=fmx_name userid=user/password@datasource">
<PARAM NAME="serverApp" VALUE="default">
</APPLET>
<!-- applet definition (end) -->

</BODY>
</HTML>
```
A.3.2 Customizing the static (non-cartridge) HTML file template

Modify the Oracle static HTML file template to fit your application as follows:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Parameter Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>codebase</td>
<td>required Replace /codebase_vdir/ with the virtual directory you defined to point to the physical directory ORACLE_HOME/forms50/java/. For example: /web_code/</td>
</tr>
<tr>
<td>archive</td>
<td>optional Replace or add to /jars_vdir/f50web.jar to provide the virtual directory path and filename of any JAR file(s) you want to be downloaded to end users’ Web browsers at application startup. For more information, refer to Section 2.2.1.4, “Configuring the Forms Client”.</td>
</tr>
<tr>
<td>serverPort</td>
<td>optional Replace 9000 with the number of the port on which the Forms Server Listener process was started. In most cases, the port number will remain 9000 (the default). For more information, refer to Section 2.2.1.2, “Starting and stopping the Forms Listener Server”.</td>
</tr>
<tr>
<td>serverArgs</td>
<td>optional Replace forms_param with any valid Form Builder command-line parameter. Replace user_param with any valid user-defined parameter.</td>
</tr>
</tbody>
</table>

Notes:
- You can provide multiple Form Builder command-line and user-defined parameters.
- You must provide a physical directory path for the .FMX file by including a directory path in the HTML file or by defining the FORMS50_PATH environment variable. The .FMX suffix is optional.

| serverApp     | optional Replace default with the name of your application class (if any). Use application classes for creating application-specific font mapping and icon path settings. For more information, refer to Section B.2, “Referencing an application class” |

*Table 6–2 Static (non-cartridge) HTML file parameters and values*
A.3.3 Examples of customized static (non-cartridge) HTML files

The following examples provide variations of the static HTML file template.

A.3.3.1 Example 1

ACME’s static HTML file (for their ORDERS application) includes the following:

- a title for the Web browser window
- a brief welcoming message to users
- a virtual directory for the Form Builder Java codebase: /web_code/
- a virtual directory for JAR files: /web_jars/
- a JAR file: f50all.jar
- two Form Builder command-line parameters: module and userid

Note: Carriage returns have been added to this example to make it more readable. When working with template files, do not add carriage returns. Refer to the online version of this file for more information.

```html
<HTML>
<!-- FILE: order.html -->

<HEAD><TITLE>ACME: Web Applications</TITLE></HEAD>

<BODY>
<CENTER><BR>Welcome to ACME Web Applications!<BR></CENTER>

<!-- applet definition (start) -->
<APPLET CODEBASE="/web_code/
    CODE="oracle.forms.uiClient.v1_4.engine.Main"
    ARCHIVE="/web_jars/f50all.jar"
    HEIGHT=20 WIDTH=20>
    <PARAM NAME="serverPort" VALUE="9000">
    <PARAM NAME="serverArgs"
        VALUE="module=c:\orant\bin\orders "userid=clerk@inventory">

    </APPLET>

<!-- applet definition (end) -->
</BODY>

</HTML>
```
A.3.3.2 Example 2

General Traders Inc. has their own static HTML file template that includes the following:

- a virtual directory for image: imag
- a background image and main image: gti_wall.jpg and gti_logo.jpg
- Form Builder command-line parameters that GTI changes for each application: module and userid

Note: Carriage returns have been added to this example to make it more readable. When working with template files, do not add carriage returns. Refer to the online version of this file for more information.

```html
<HTML>
<!-- FILE: gti_stat.html -->
<!-- GTI static HTML template -->
<!-- Rename and modify as needed -->

<BODY BACKGROUND="/imag/gti_wall.jpg">
<CENTER><IMG SRC="/imag/gti_logo.gif"

HEIGHT=170
WIDTH=120
ALT="gti_logo.jpg">

</CENTER>

</BODY>

</HTML>
```
A.3 Working with the Static (Non-Cartridge) HTML File Template
This appendix covers these topics:

- Section B.1, “About Application Classes”
- Section B.2, “Referencing an application class”
- Section B.3, “Setting an icon directory path for an application class”
- Section B.4, “Creating font-mapping settings for an application class”

## B.1 About Application Classes

Application classes enable you to specify the following custom settings for your Web-enabled Forms applications:

- the directory location of images files (in GIF format) for iconic buttons
- font mappings (application fonts to Java fonts)

To create settings for an application class, you must add the appropriate settings to the REGISTRY.DAT file:

```
ORACLE_HOME\forms50\java\oracle\forms\uiClient\v1_4\util\Registry.dat
```

Using application classes is optional, since Form Builder supplies default settings for font mappings and icon path. Use application classes if you wish to create a number of different settings that you can alternate between by switching a single reference in your application’s cartridge settings, cartridge HTML file, or static HTML file.

At runtime, Form Builder first looks for settings that correspond to the specified application class (if any). If no application class settings exist, Form Builder uses default settings.
B.2 Referencing an application class

To reference an application class for your application:

1. Include the serverApp parameter (and value) in your application’s cartridge settings, cartridge HTML file, or static HTML file.

   For example:

   ```
   <applet codebase="/web_code/"
   code="oracle.forms.uiClient.vl_4.engine.Runform"
   height="20" width="20">
   <param name="serverPort" value="9000">
   <param name="serverArgs" value="Module=inventory">
   <param name="serverApp" value="inv_form">
   </applet>
   ```

   Note: Carriage returns have been added to this example to make it more readable. When working with template files, do not add carriage returns.

2. Add font mapping and icon path settings (in the REGISTRY.DAT file) specific to the application.

   For more information refer to:
   - Section B.3, “Setting an icon directory path for an application class”
   - Section B.4, “Creating font-mapping settings for an application class”
B.3 Setting an icon directory path for an application class

To set an icon directory path for an application class:

1. Open the REGISTRY.DAT file in a text editor.
2. Add the following line to REGISTRY.DAT:

   <app_class>.iconpath=<path>

Replace <app_class> with a valid application class referenced by your application, and replace <path> with one of the following:

- a fully-qualified URL that points to any virtual directory on any server:
  
  inv_form.iconpath=http://abc_dev.com/html/icons/

- a virtual directory on the same server from which the HTML page is downloaded:
  
  inv_form.iconpath=/icons/

- a directory relative to the directory from which the HTML page was downloaded (non-cartridge implementations only):
  
  inv_form.iconpath=icons/

**Note:** If you do not specify an icon path for an application class, Form Builder uses the default icon path setting in REGISTRY.DAT:

  default.iconpath=

By default, Form Builder will search for iconic button image files in the directory from which the HTML file was downloaded.

To specify the directory Oracle will search by default, append it to the Form Builder icon path setting. The path guidelines provided above still apply. For example:

  default.iconpath=http://abc_dev.com/html/icons/
B.4 Creating font-mapping settings for an application class

To create font-mapping settings for an application class:

1. Open the REGISTRY.DAT file in a text editor.
2. Add lines to the file as follows:
   
   `<app_class>.<font_param>=<param_value>`

   Replace `<app_class>` with a valid application class, replace `<font_param>` with the name of the font parameter, and replace `<param_value>` with an appropriate value.

To map fonts from your application to their Java equivalents, add lines similar to:

   `inv_form.appFontnames=Courier,Terminal,Arial`
   `inv_form.javaFontnames=MonoSpaced,Dialog,SanSerif`

To map unrecognized application fonts to a Java font, add lines similar to:

   `inv_form.defaultFontname=Dialog`
   `inv_form.defaultSize=10`
   `inv_form.defaultStyle=BOLD`

Notes:

- The mapping of Java fonts to platform-specific fonts is automatic, according to default Java settings; you cannot alter the mappings. For more information, refer to Section 2.3.2.4, “Selecting fonts”.
- For more information on setting font preferences, refer to the commented sections of the REGISTRY.DAT file.
**Glossary**

**action**
In Project Builder, a command string supplied either by Project Builder or by the user that applies to files of a given type or types. An action is not restricted to a single file type; for example, if the action “Compile” is defined for both forms and C source files, selecting the menu item **Compile Project** will compile all .FMB and .C files using the appropriate tools. See also: *pre-defined action, user-defined action.*

**applet**
A Java term for small programs that can be dynamically imported into Web pages or applications as needed.

**bidirectional support**
Support for languages whose natural writing direction is right-to-left, for example Middle Eastern and North African languages.

**block**
The representation of an entity on a form.

**built-in macro**
In Project Builder, a macro shipped with Project Builder. See also: *macro.*

**canvas**
The surface on which interface items and prompts are drawn. Canvasses are displayed in a window.
CGI — Common Gateway Interface
The industry-standard technique for running applications on a Web server. Whereas standard HTML documents retrieved from a Web server are static (the exact same text is retrieved every time) CGI enables a program running on the Web server to communicate with another computer to generate “dynamic” HTML documents in response to user-entered information.

dependency view
In Project Builder, a view that shows the files in the Project Navigator in the order in which they depend on each other, with project nodes at the highest point in the hierarchy, followed by target nodes, which are followed by buildable components of those targets. For example, an executable form depends on and will be followed by an .fmb file, which may depend on and be followed by a library used for a USER-EXIT procedure, and so on. See also: project view, target.

dialog box
A window used to enter information needed to complete a specific action. The user must interact with this window before proceeding.

encryption
The practice of scrambling (encrypting) data in such a way that only an intended recipient can unscramble (decrypt) and read the data.

entity
A thing of significance to the user. ‘Assignments’ and ‘Sales Order Lines’ are examples of entities. A single entity may comprise several blocks, such as ‘Sales Rep’, ‘Quotas’, and ‘Territories’.

character set
Encoding scheme in which each character is represented by a different binary value. For example, ISO8859-1 is an extended Latin character set that supports more than 40 Western European languages.

deliver
In Project Builder, to prepare and provide a completed application for distribution and deployment.

export
In Project Builder, the process of writing out a file containing project, type, action, and/or macro definitions in a portable format for distribution to others who may work on heterogeneous platforms. See also: export file, import.

export file
In Project Builder, the shareable, portable file created by exporting a project. The default extension of an export file is .UPX. See also: export, import.

field
An interface element that displays information to the user and/or accepts input from the user. Text items, check boxes, and poplists are examples of fields. Also known as ‘widget’ or ‘item’.

firewall
A computer that regulates access to computers on a local area network from outside, and regulates access to outside computers from within the local area network.

format mask
A setting that defines the appearance of the value of a field. For example, a format mask is used to specify the display of currency amounts and dates.

Global Registry
A Project Builder registry that stores information common to an entire Developer/2000 installation. This information is restricted to type definitions and their associated actions and pre-defined or user-defined properties. The use of the Global Registry is optional; its functions can be performed by individual user registries. See also: registry, user registry.

group
In Project Builder, collections of related items available via submenus off the Launcher. Groups enable users to set up the Launcher much like the Windows 95 Start menu, with arbitrary “groups” that pop up to reveal other items and/or groups.
GUI — Graphical User Interface
The use of pictures rather than just words to represent the input and output of a program. Programs with GUls run under a windowing system (such as X Windows, Microsoft Windows, Apple Macintosh, and so on). GUI programs display icons, buttons, and so on, in windows on the screen; users control the GUI programs mainly by moving a pointer on the screen (typically controlled by a mouse).

HTML — Hypertext Markup Language
A tag-based ASCII language used to specify the content and hypertext links to other documents on WWW servers on the Internet. End users with Web browsers view HTML documents and follow links to display other documents.

HTTP — Hypertext Transfer Protocol
The protocol used to carry WWW traffic between a WWW browser computer and the WWW server being accessed.

hyperlink
A reference (link) from some point in one hypertext document to (some point in) another document or another place in the same document. A Web browser usually displays a hyperlink in some distinguishing way (in a different color, font or style). When users activate hyperlinks (by clicking on them with a mouse) the browser displays the target of the link.

hypertext
A collection of documents containing cross-references which, with the aid of a Web browser, allow readers to move easily from one document to another.

implied item
In Project Builder, a project item, usually the result of automatic generation, which Project Builder recognizes and for which it automatically creates an entry in the Project Navigator. For example, Project Builder can recognize .OBJ files, generated as an immediate step in the compilation of C source files, and create entries for them in the Project Navigator. Although resetting the properties of an implied item is of limited use (the next compilation will destroy changes) such items can be useful, as they can be examined via actions such as Edit, View, and Print. See also: action, item.

import
In Project Builder, to read in a file containing project information. This is the recommended method for sharing projects. See also: export, export file.
**Inherit**
In Project Builder, to obtain information for an action, type, macro, or property definition from an ancestor node in the dependency tree. If related attributes exist in an ancestor node, they may be inherited. Thus, filesystem items like forms and documents may inherit action definitions from subprojects, projects, a user registry, or the Global Registry; projects may inherit type definitions from a user registry or the Global Registry; and so on.

**Input Item**
In Project Builder, the file used to build a target. For example, an .FMB is the input item for an .FMX. Also called *source*.

**Internet**
A worldwide TCP/IP-based network of computers.

**Intranet**
An internal TCP/IP network, access to which is restricted (via a firewall) to individuals inside the company or organization. An intranet provides similar services within an organization to those provided by the Internet, but is not necessarily connected to the Internet. A common example of an intranet is when a company sets up one or more Web servers on an internal network for distribution of information or applications within the company.

**IP (Internet Protocol) Address**
A four-part number with no more than three digits in each part that uniquely identifies a computer on the Internet.

**Item**
In Project Builder, an object in the file system associated with a project, such as a form or report, and pointed to or represented by a node in the Project Navigator.

**JAR — Java ARchive**
A file used for aggregating many files (Java class files, images, and so on) into one file.

**Java**
A computer language that supports programming for the Internet in the form of platform-independent “applets”. 
language environment variable
Environment variable which specifies the language, territory, and character set for a user’s environment. The language environment variable can be any one of the following: NLS_LANG, DEVELOPER_NLS_LANG, or USER_NLS_LANG.

Launcher
In Project Builder, the secondary toolbar docked (by default) to the left of the Project Navigator. It provides simple organizational and application launching abilities.

macro
In Project Builder, a type-specific variable which may be used to customize and extend actions. A macro may be either a constant or a simple expression (which may, in turn, contain other constants and/or expressions). The use of macros offers great flexibility in issuing command options, and in allowing the user to modify sets of commands by changing one property definition.

master-detail
A relation between two entities that indicates a hierarchy of information. For example, a Sales Order consists of a Header entity and a Line entity; the Header is the master of the Line, and the Line is the detail of the Header.

modal
A state where the user must supply specific information before continuing operation of the application.

multilingual application
An application which can be deployed in more than one language and displays data according to local conventions.

ORACLE_HOME
An environment variable that indicates the root of the Oracle7 Server code tree.

PDF — Portable Document Format
A file format (native for Adobe Acrobat) for representing documents in a manner that is independent of the original application software, hardware, and operating system used to create the documents. A PDF file can describe documents containing any combination of text, graphics, and images in a device-independent and resolution independent format.
**PL/SQL**

Oracle’s proprietary extension to the SQL language. Adds procedural and other constructs to SQL that make it suitable for writing applications.

**port**

A number that TCP uses to route transmitted data to and from a particular program.

**pre-defined action**

An action shipped with Project Builder and automatically available to the user via a menu item and/or a toolbar button. Pre-defined actions include Build, Deliver, and several source control options. When a pre-defined action is defined for a supported file type, the action is invoked for any selected item of that file type when the user calls that action from Project Builder. See also: *action, user-defined action*.

**project**

The basic data structure created by Project Builder. A project is a collection of pointers to files in the user’s file system. It also contains information about behavior that the user may wish to apply to a given project, such as the specific editor to invoke to edit all files of types .CPP, .H, and .TXT. Project files can be exported and shared across platforms. See also: *export, project definition file, project item*.

**project definition file**

In Project Builder, a file that stores project data, which consists of project items and their properties. Each file has one project item by default, which can be thought of as the “root” or master project for that file. The user can create as many subproject items as necessary in this file; subprojects are items beneath the master project which allow the user to collect subgroups of items and change their properties at the parent (subproject) level. The default extension for a project file is .UPD. See also: *project, project item*.

**project item**

In Project Builder, an item that stores project-specific information, such as a connection string and an implicit list of contained items. Types are not defined here, but actions and user-defined macros for all types visible to a project may be defined and/or overridden here. See also: *item, project, project definition file*. 
**Project Navigator**
In Project Builder, the window containing a hierarchical list of project items for the current session. The list appears in outline form, and enables the user to complete several tasks, such as creating, editing, renaming, and deleting objects. Although only one schema is visible at any time, the user can choose from two different schema by which to organize the objects. See also: dependency view, project view.

**project view**
In Project Builder, the project view shows objects in the Project Navigator organized by their type, and by their project/subproject relationships. The projects are organized alphabetically by project file, then alphabetically by category. See also: dependency view, Project Navigator.

**Project Wizard**
In Project Builder, a dialog that assists the user in accomplishing the steps necessary to create a new project or subproject.

**prompt**
A label that uniquely identifies an item. ‘Salesperson’ and ‘Item Description’ are examples of prompts.

**region**
A set of related items within an entity. For example, the Purchase Order Header entity might contain a ‘Currency Information’ region, which consists of the Rate, Type, and Date fields.

**registry**
In Project Builder, a global and/or user-specific configuration file used to store project definitions and environment information. See also: Global Registry, user registry.

**RDBMS — Relational Database Management System**
A database that allows the definition of data structures, storage and retrieval operations, and integrity constraints. In such a database, data and relations between them are organized in tables.

**snap point**
The point of a widget that corresponds to the (X,Y) position that locates it.
**socket**
The combination of an IP address and a port number.

**target**
In Project Builder, any item in the middle of the dependency tree; for example, an executable is a (compile) target for a library, while a library is a target for a group of objects and an object is a target for a source file. See also: *input item*.

**toolbar**
A series of iconic buttons that perform generic actions, such as List and Save.

**TCP — Transmission Control Protocol**
The underlying communication protocol for exchanging HTTP requests between clients and Web servers.

**type**
In Project Builder, a description of a file type, such as a form, a document, etc., containing such information as type name and description. Types are the foundation for defining actions and macros.

**URL: Uniform Resource Locator**
The “address” used to specify a WWW server and home page. For example:

```
http://www.acme.com/
```

indicates that the host’s address is *www.acme.com*.

An URL most often is a filename (possibly with a long path to it and usually with an extension of `.HTML`, or `.HTM` (for PC-DOS filenames).

**user-defined action**
In Project Builder, a custom action defined by a Project Builder user. Such actions may apply to a single file type, or all file types. See also: *action, pre-defined action*.

**user-defined macro**
In Project Builder, a custom macro defined by a Project Builder user. Such macros may be used to modify both pre-defined and user-defined actions. See also: *action, built-in macro, macro, pre-defined action*.
user registry
In Project Builder, a project file storing configuration information on a per-user basis. This enables users to customize their individual development environments. The user registry inherits type information from the Global Registry, and may define new types as well as override aspects of types defined in the Global Registry. It also stores environment and preference information, such as the user’s preferred connection string, UI settings, and general project information. See also: Global Registry, registry.

virtual directory
A synonym that the virtual file system maps to a file stored in the file system maintained by the host machine’s operating system.

virtual file system
A mapping that associates the pathnames used in URL to the file system maintained by the host machine’s operating system.

Web browser
A program that end users utilize to read HTML documents and programs stored on a computer (serviced by a Web server).

Web cartridge
A program executed on a Web server via the WRB.

Web server
A server process (HTTP daemon) running at a Web site which sends out Web pages in response to HTTP requests from remote Web browsers.

window
A screen in a graphical user interface (GUI) environment. A window is a frame enclosing a surface on which elements are painted.

WRB — Oracle Web Request Broker
Provides a powerful distributed runtime environment for developing and deploying applications for the Web. The WRB runtime platform enables application developers to write applications that are independent of, and work with a number of, Web servers.
WWW — World Wide Web
The network of servers on the Internet, each of which has one or more home pages, which provide information and hypertext links to other documents on that and (usually) other servers.
Index

A

action
  in Project Builder
    automating, 1-25
    definition, 1-22
    multiple platforms, 1-40
  in Web report, 2-149
ActiveX controls
  built-ins, 6-244
  examples, 6-260
  properties, 6-256
  use guidelines, 6-257
  using within Developer/2000, 6-254
alerts, 3-185
ALTER SESSION, 4-202
  using to change NLS_LANG, 4-199
  using to specify default format mask, 4-202
anchor, 3-191, 3-192
application
  associating modules with, 1-25
  customizing using foreign functions, 6-263
  deploying, 1-43
  design and development, 1-28
  designing for portability, 5-213
  designing user interface, 3-159
  maintenance and support, 1-36
  managing, 1-19
  multilingual, 4-199
  multiple platforms, 1-39
  project administrator role, 1-37
  release phase, 1-42
  running against ODBC datasources, 6-308
  software development life cycle, 1-20
test phase, 1-40
  translating, 4-210
  Web design guidelines, 2-137
application cartridge HTML files
  examples, A-319
  Oracle template, about, A-316
  Oracle template, modifying, A-318
  Oracle template, snapshot of, A-317
  setting parameter values in, A-318
  specifying as a cartridge parameter, 2-81
application class
  creating, B-327
  creating font-mapping settings, B-330
  setting icon directory path, B-329
application classes
  about, B-327

B

bar graph, 3-196
Base Printing On property, 3-193
bidirectional support, 4-208
  in Form Builder, 4-208
  in Report Builder, 4-209
  language environment variable, 4-208
Big Font, 4-205
  block
    in Form Builder
      definition, 3-169
      design guidelines, 3-181
    boilerplate, 3-191
    Build From type action, 1-26
    built-ins
      OLE and ActiveX, 6-244
button, 3-191
  portability considerations, 5-219, 5-231

canvas
  in Form Builder
    definition, 3-172
    design guidelines, 3-177
cartridge HTML file, A-316
cartridge parameters
  REPORTS30_OWSDIAGBODYTAGS, 2-116
  REPORTS30_OWSDIAGHEADTAGS, 2-116
  REPORTS30_OWSHELP, 2-116
  REPORTS30_OWSMAP, 2-116
  REPORTS30_OWSNODIAG, 2-117
  REPORTS30_OWSPATHONLYURL, 2-117

CGI (Common Gateway Interface)
  configuring for Report Builder, 2-110
  dynamic reporting with, 2-64
  environment variables for Report Builder, 2-114
  specifying URL report run requests, 2-119

Chapter Title, 6-235

color
  design guidelines, 3-176
  portability considerations, 5-216, 5-231

Common Gateway Interface See CGI

compiling
  modifying results, 1-39
  project, 1-38

connection strings
  creating, 1-32

console
  portability considerations, 5-221
  container window, 3-170

context-sensitive help, 3-187

d Control block, 3-169

CT ChapterTitle, 6-235

D

data block, 3-169
Data Model view, 3-190

default format mask
  specifying with ALTER SESSION, 4-202
  specifying with the language environment variable, 4-201

DEI file, 1-47
Deliver File property, 1-27
Deliverable Type property, 1-26
DEPLOY directory, 1-47
Designer/2000
  using with Developer/2000, 2-141
  DEVELOPER_NLS_LANG, 4-199, 4-200
  obtaining the current value of, 4-201
  using for bidirectional applications, 4-208
distribution media, 1-44
  definition of, 1-44

Double-Y, 3-197
drill-down reporting, 2-56
dynamic reporting, 2-56

E

embedded object, 6-238
encryption
  Web applications, 2-62
entry
  in Project Builder, 1-22

environment variables
  LD_LIBRARY_PATH, 2-115
  ORACLE_HOME, 2-114
  REPORTS30_CGIDIAGBODYTAGS, 2-118
  REPORTS30_CGIDIAGHEADTAGS, 2-118
  REPORTS30_CGIHELP, 2-117
  REPORTS30_CGIMAP, 2-118
  REPORTS30_CGINODIAG, 2-118
explicit anchor, 3-192
export
cross-platform development, 1-27
external activation, 6-239

F
field, 3-191
font
creating font-mapping settings, B-330
portability considerations, 5-217, 5-230
font aliases, 5-218
font aliasing, 4-206
font substitution, 4-206
font support
for Unicode, 4-205
fonts, 2-142
in HTML report output, 2-152
mapping application fonts to Java fonts, B-330
foreign function
creating, 6-268
e examples, 6-276
interface, 6-265
use guidelines, 6-266
using within Developer/2000, 6-263
form
character-mode platforms, 5-227
Form Builder
application-specific settings, B-327
bidirectional support, 4-208
building effective forms, 3-168
character-mode platforms, 5-227
design guidelines, 3-174
designing for portability, 5-214, 5-224
using with Open API, 6-290
Web applications
architecture, 2-57
configuring, 2-74
design guidelines, 2-140
troubleshooting, 2-97
Form module, 3-168
format element
number, 4-202
format mask
default, 4-201
design considerations, 4-204
specifying default with ALTER SESSION, 4-202
specifying default with the language environment variable, 4-201
Forms Cartridge Handler, 2-78
about, A-316
Forms Client, 2-57
Forms Generate component, 2-74
Forms Server, 2-58
Forms Server Listener, 2-75
Forms Sever Runtime Engine, 2-58
FORMS50_MAPPING, 2-138
FORMS50_OUTPUT, 2-138
FORMS50_REPFORMAT, 2-138
frame, 3-191
in Form Builder, 3-170

G
Gantt, 3-197
get_application_property, 5-226
Global Registry, 1-29
in Project Builder, 1-23
Graphics Builder
creating effective displays, 3-195
designing for portability, 5-232
running from Form Builder, 2-139
Web applications
architecture, 2-68
configuring, 2-131
design guidelines, 2-155
Graphics Client, 2-68
Graphics Server, 2-69
GTM GlossaryTerm, Glossary-340
GUI (graphical user interface)
see user interface
Index-348

GW_IMAGES_USE_FILE, 2-134
GW_LINKS_CLOSE_PREV, 2-135
GW_WRITE_TRACE_FILE, 2-135

H

headings
  H1 Head1, 3-189, , 3-190, , 3-191, , 6-236
  H2 Head2, 6-236
High-low, 3-197
Horizontal Elasticity, 3-193
HTML (Hypertext Markup Language)
  application files (non-cartridge) See non-cartridge application HTML files
    report output, 2-151
    template files, A-315
    template HTML files, about, A-315
  HTML file, 2-94
HTTP (Hypertext Transfer Protocol)
  redirection, report output and, 2-67

I

icon
  portability considerations, 5-219
iconic buttons
  application classes and, B-327
directory paths, B-327
  images formats, B-327
images
  in HTML report output, 2-153
implicit anchor, 3-192
implied item, 1-26
import
  cross-platform development, 1-27
in-place activation, 6-239
INS file, 1-47
installable component, 1-44
installation
  files, 1-45
  process, 1-48
Internet, 2-54
intranet, 2-54
item
  in Form Builder

definition, 3-169
design guidelines, 3-181

J

JAR files, 2-85
Java applet, 2-57
JAVA fonts, 2-142

K

Kanji characters, 4-208
Keep with Anchoring Object property, 3-195

L

language environment variable, 4-199
  DEVELOPER_NLS_LANG, 4-199
  NLS_LANG, 4-199
  USER_NLS_LANG, 4-199
  using to specify character set, 4-204
  using to specify default format mask, 4-201
Launcher, 1-23
Launcher toolbar, 1-27
Layout Model Objects, 3-191
Layout Model view, 3-190, , 3-191
Line chart, 3-197
linked object, 6-238
load balancing
  cartridge base HTML file, 2-94
  example, 2-87
  Metrics Server and Client installation, 2-93
  requirements, 2-90
  setting up, 2-87
  web cartridge configuration, 2-91
LRS
  See Log roll-forward server (LRS), A-315

M

macro
  in Project Builder, 1-22
  multiple platforms, 1-40
MAP file, 1-46
mapping
application fonts to Java fonts, B-330

menu
in Form Builder
design guidelines, 3-188
portability considerations, 5-220

Menu module, 3-168
messages
in Form Builder
design guidelines, 3-185

Metrics Client
installation, 2-93
starting, 2-95

Metrics Server
installation, 2-93
starting, 2-94
microhelp, 3-187
moat, 5-219
modal window, 3-170
modeless window, 3-170
modifying
Oracle application cartridge HTML file
template, A-318
Oracle non-cartridge application HTML file
template, A-323

modules
adding to project, 1-32
assigning connection strings to, 1-26
checking in and out, 1-39
creating dependencies, 1-25, 1-32
creating install package, 1-27
editing, 1-38
in Form Builder, 3-168
monitor
portability considerations, 5-216
multibyte character set, 4-199, 4-205
multilingual application, 4-199
translating, 4-210
multiple datasources
See also OCA (Open Client Adaper)
use guidelines

N

National Language Support (NLS), 4-199
Network Computing Architecture (NCA), 2-56

NLS, see National Language Support

NLS_LANG, 4-199
changing with ALTER SESSION, 4-199
setting for Unicode, 4-206
setting for UTF-8, 4-206
syntax, 4-199

NLSSORT, 4-207
non-cartridge application HTML files
examples, A-324
Oracle template, about, A-321
Oracle template, modifying, A-323
Oracle template, snapshot of, A-322

non-cartridge HTML file template
See static HTML file template
non-Oracle HTML file template
non-Oracle foreign function, 6-264
number format element, 4-202

O

object group, 3-164
object library, 3-163
definition, 3-173
Object Library module, 3-169

OCA
See OCA (Open Client Adapter)
OCA (Open Client Adapter)
OCA.PLL, 6-310
overview, 6-308
running applications against ODBC
datasources, 6-312
use guidelines

OCX
See ActiveX controls

ODBC (Open Database Connectivity)
See OCA (Open Client Adapter)

OLE (Object Linking and Embedding)
about OLE automation, 6-240
about OLE servers and containers, 6-237
built-ins, 6-244
container properties, 6-242
embedded objects, 6-238
examples, 6-251
external activation, 6-239
in-place activation, 6-239
linked objects, 6-238
registration database, 6-238
See also ActiveX controls
use guidelines, 6-249
using within Developer/2000, 6-236
online help
implementing, 3-187
portability considerations, 5-223
Open API
creating or modifying modules, 6-293
examples, 6-294
overview, 6-290
use guidelines, 6-293
Open Client Adapter
See OCA (Open Client Adapter)
OPENDB.PLL, 6-310
operating system
portability considerations, 5-222
ORA_FFI, 6-265
Oracle Applications object library, 3-163, 3-173
Oracle Call Interface foreign function, 6-264
Oracle File Packager, 1-43, 1-44
Oracle Installer, 1-43, 1-45
Oracle precompiler foreign function, 6-264
Oracle Web Request Broker, 2-69, 2-78
ORACONNECT, 1-22

P
Page Break After property, 3-194
Page Break Before property, 3-194
Page Protect property, 3-195
Parameter Form view, 3-190
parameters
GW_IMAGES_USE_FILE, 2-134
GW_LINKS_CLOSE_PREV, 2-135
GW_TIMEOUT, 2-133
GW_WRITE_TRACE_FILE, 2-135
PDF, 4-205
PDF (Portable Document Format), 2-154
pie chart, 3-196
PJ10
Project Builder environment variable, 1-29
PL/SQL
translating strings, 4-211
PL/SQL libraries
using to translate a multilingual application, 4-211
PL/SQL Library module, 3-168, 3-189
platform
portability considerations, 5-222
popup hints, 3-187
port
specifying as a cartridge parameter, 2-82
portability
designing applications, 5-213
managing multi-platform projects, 1-39
registries, 1-39
user interface considerations, 3-165
PRD file, 1-45
preface
Send Us Your Comments, xi
Print Object On property, 3-193
Product
definition of, 1-44
project
building, 1-38
creating, 1-30
definition, 1-22
multiple platforms, 1-39
packaging for release, 1-43
project administrator, 1-24
creating a project, 1-30
definition of role, 1-24
managing multi-platform projects, 1-40
release phase, 1-43
test phase, 1-41
working with projects, 1-37
Project Builder
accessing other tools, 1-27
benefits, 1-25
installing, 1-29
overview, 1-21
roles, 1-24
terminology, 1-22
project items
definition, 1-23
implied items, 1-26
Project Navigator, 1-23
project registry file
definition, 1-23
sharing and porting, 1-27
Project Wizard, 1-30
prompt
portability considerations, 5-221
Property Palette, 1-23
PVCS, 1-28

R

RDF files
deploying, 2-125
region
in Form Builder
definition, 3-170
design guidelines, 3-180
registration database, 6-238
registry
in Project Builder, 1-23
portability, 1-39
registry file
sharing and porting, 1-27
REGISTRY.DAT file
about, B-327
setting icon directory path in, B-329
release phase, 1-42
repeating frame, 3-191
report
on character-mode platforms, 5-231
Report Builder
bidirectional support, 4-209
building effective reports, 3-188
controlling layout objects, 3-191
designing for portability, 5-230
Editor views, 3-190
escapes, 2-149
HTML output, about, 2-151
modules, 3-189
output format, 2-150
templates, 3-190
tips, 2-155
Web applications
architecture, 2-63
configuring, 2-97
design guidelines, 2-144
environment variables, 2-114
troubleshooting, 2-126
Report Editor Views, 3-190
Report module, 3-189
reports
actions, about, 2-149
adding Web functionality, 2-145
addresses, in HTML output, 2-152
bookmarks, about, 2-148
bookmarks, in HTML output, 2-152
buttons, in HTML output, 2-152
buttons, in PDF output, 2-155
drawings, in HTML output, 2-152
escapes, about, 2-149
fonts, in HTML output, 2-152
frames, in HTML output, 2-152
hyperlinks, about, 2-145
images, in HTML output, 2-153
OLE objects, in HTML output, 2-153
OLE objects, in PDF output, 2-155
overlapping objects, in HTML output, 2-153
pagination, in HTML output, 2-153
PDF output, about, 2-154
Reports Queue Manager, 2-57
Reports Server
about, 2-65
configuration file, 2-105
configuring, 2-100
Reports Web Cartridge
about, 2-65
configuring, 2-110
dynamic reporting with, 2-64
environment variables, 2-114
specifying URL report run requests, 2-119
Reports Web CGI
about, 2-66
reusable components
for portability, 5-221
RSA RC4 40 bit encryption, 2-62
RUN_PRODUCT, 2-137

S
Scatter, 3-197
screen design considerations, 4-207
screen design
for translation, 4-207
Send Us Your Comments boilerplate, xi
session IDs, 2-69
setting
parameter values for application cartridge
HTML files, A-318
settings
for Web-enabled Form Builder applications, B-328
SHOW_DOCUMENT, 2-142
software development life cycle, 1-20
sorting character data, 4-207
source control
multiple platforms, 1-40
setting up, 1-33
using with Developer/2000, 1-28
specifying
cartridge HTML file as a cartridge parameter, 2-81
port as a cartridge parameter, 2-82
SQL*Net SNS/ANO, 2-62
Stacked canvas, 3-172
Stage area
definition of, 1-44
stage area, 1-44
Standard object library, 3-163, 3-173
StarBase, 1-28
StarTeam, 1-28
static HTML file template
storyboard, 3-165
subproject
in Project Builder, 1-22
system test, 1-20

Tab canvas, 3-172
template, 3-164, 3-190
template HTML files, A-315
templates
HTML file templates See application cartridge
HTML files and Non-cartridge application
HTML files
See application cartridge HTML files
See Non-cartridge application HTML files
test phase, 1-40
thin client, 2-56
three-tiered architecture, 2-54
Toolbar canvas, 3-172
Tooltips, 3-187
translating a multilingual application, 4-210
using PL/SQL libraries, 4-211
using Translation Builder, 4-210
translating the Designer, 4-210
Translation Builder, 4-210
using to translate a multilingual application, 4-210
type
in Project Builder, 1-22

U
UCS-2, 4-205
UIFONT.ALI, 5-218
Unicode, 4-205
font support, 4-205
setting NLS_LANG, 4-206
support in Developer/2000, 4-205
UCS-2, 4-205
UTF-7, 4-205
UTF-8, 4-205
unit test, 1-20
URL
parameterized, 2-71
specifying report run requests, 2-119
URL (Uniform Resource Locator)
exmple report run requests, 2-119
mapping for report run requests, 2-122
user exit
interface to foreign functions, 6-265, 6-273
ORA_FFI, 6-265
portability considerations, 5-224
user feedback
gathering, 3-167
user interface
building, 3-167
designing, 3-159
    designing for portability, 5-215
translating, 4-210
user registry
    in Project Builder
customizing, 1-35
definition, 1-23
user requirements
    defining, 3-161
USER_NLS_LANG, 4-199, 4-200
    obtaining the current value of, 4-201
    using for bidirectional applications, 4-208
UTF-7, 4-205
UTF-8, 4-205
    setting NLS_LANG, 4-206

V
version label, 1-37
versions
    synchronizing, 1-37
Vertical Elasticity, 3-193
viewports, 3-172
virtual directories, 2-77
VRF file, 1-47

W
web applications
    classes See application classes
Web cartridges, 2-56
Web publishing, 2-56
white space, 4-207
window
    in Form Builder
definition, 3-170
design guidelines, 3-179
World Wide Web
    architecture
    Form Builder, 2-57
    Graphics Builder, 2-68
    Report Builder, 2-63
configuring for Web applications, 2-74
deploying applications
    Form Builder, 2-61

overview, 2-54
designing web applications, 2-137
troubleshooting
    Form Builder, 2-97
WRB (Web Request Broker)
    creating application cartridges with, 2-79, 2-110, 2-131