1

Putting Together a NEXTSTEP Application

There are a number of ways you might draw the line between programs and applications. Programs are simple; applications are complicated. Programs are small; applications are big. Programs run from a command line; applications have a graphic user interface. A program has just a few source files; an application may have lots and lots.

No matter how you draw the line, as you move from writing programs to developing applications, you need to focus increasing attention on project management. If the application is the end result, the project is how you get there. The project can be thought of as both the steps you go through and the source files you use to construct an application.

A complete project management strategy includes strategies for creating, organizing, and maintaining source files, building the application from its sources, running and debugging the application, revising the source files to fix bugs, and installing the finished applicationÐor preparing it for others to install.

In NEXTSTEP, the hub of application development is Project BuilderÐa project manager that is itself a NEXTSTEP application. Project Builder isn't the only tool you use to manage your project and develop your application. Instead, it's like a control center from which you switch from one application development task to another, and from one tool to another.

This chapter takes a brief look at the components of a NEXTSTEP application. It explains the path that Project Builder and other NEXTSTEP tools offer you for going from a set of source files to a working application. It looks at the application development process in terms of resources and tasks that you, the developer, must provide and those that Project Builder and other NEXTSTEP tools provide for you. Subsequent chapters present detailed reference for each of the tools introduced here. The last four chapters present step-by-step tutorials that offer you a chance to explore the NEXTSTEP development process for yourself.

The Application Development Process

The process of developing an application can be divided into three general tasks: designing, coding, and debugging. These tasks are never performed entirely sequentially. You may decide after some coding that you need to change some aspect of design. Debugging always reveals code that needs rewriting, and occasionally exposes design flaws. When you develop an application with NEXTSTEP, you can move easily among these tasks.

The following sections enumerate the components of the NEXTSTEP application development process, describing those portions for which you're responsible and those which Project Builder,

Interface Builder, and other NEXTSTEP development tools handle for you. For more information on Project Builder, see Chapter 2; for more on Interface Builder, see Chapter 3.

Design Your Application

Before you write any code, you should spend some time thinking about design. Some components of application design to consider are functionality, program structure, and user interface. You should think about the goals of your application and the techniques you might use to meet those goals. You should determine the unique classes that your application will require and think about how to divide your program into separate modules. You should sketch out user interface ideas, and use Interface Builder to prototype and test those ideas.

Create a Project

With the basic design determined, you can use Project Builder to start a new project.

In NEXTSTEP, a project is physically represented by a directory under the control of Project Builder; all of the components of the project must reside in this directory. When you start a new project, Project Builder automatically generates the project directory and a set of source files common to all applications, including a main file, a nib file, a makefile, and others. The main file includes the standard **main()** function required in all C programs. The nib file is used by Interface Builder to archive the application's user interface (nib is an acronym for ªNEXTSTEP Interface Builderº). The makefile is updated by Project Builder to keep track of all the source files from which your application is built. Another file in the project directory, **PB.project**, is used by Project Builder itself to keep track of various project components.

Throughout the life of the project, you will add to and update the files in the project directory. NEXTSTEP development tools, including Project Builder and Interface Builder may add to and maintain other files in this directory as your project grows.

Write Code for Your Application

To establish the unique workings of your application, you create class interface and implementation files that include code for the appropriate methods and instance variables. Interface Builder can help in this process by creating skeletal code for a class if you list the methods in the Inspector panel. If you create the source files first, Interface Builder can parse them to learn about their **id** instance variables and action methods.

Project Builder lets you add source files to your project at any time. You can create other source files using standard C, Objective C, and C++ code. Project Builder can also know about and manage other files, such as **pswrap** files containing PostScript code within C function wrappers.

Connect Objects with Interface Builder

In Interface Builder, you can interconnect objects in your application. For example, you can establish the target and action for a control in the interface.

Interface Builder puts information about the classes used by your application in the nib file; included are Application Kit classes and other classes provided by NEXTSTEP, as well as the custom classes you define. The nib file contains all the information required to generate the objects in your application at run time: specifications for objects, connections between objects, icons, sounds, and

other features. A NEXTSTEP application can have one or more nib files for each application you create.

Add Other Resource Files

Resource files are frequently used to customize the user interface for your application. Project Builder allows you to add icons for both your application and its documents. Interface Builder allows you to add icons and sounds for the buttons in your user interface. You can put other images in your application using Application Kit classes and PostScript code. You can add other sounds using Sound Kit™methods. Project Builder provides a drag-and-drop interface for adding sounds, images, and other resource files to your project, including unique icons for your application and its document files.

Choose Document Extensions for Your Application

If your application reads and writes documents, you'll need to take measures to see to it that the Workspace Manager™knows about and can work with those files. First, you need to write file management code that saves the documents with a unique extension. You also need to use the Project Builder application's Attributes display to specify document extensions for an application. Project Builder adds these extensions to the appropriate file to assure that your application is invoked by Workspace Manager when the user double-clicks a file with the specified extensions.

If you plan to distribute your software, or want to avoid future collisions with file extensions used by other applications, register the document file extensions with the NeXT Extension Registry. A list of currently registered names and the address for the extension registry is included in the *User Interface Guidelines*.

Compile Your Program

As you add source files to your application, Project Builder lists them in the project makefile. When you use its Build command, Project Builder starts the **make** program which in turn reads the project makefile and generates the executable file from the sources. As **make** runs, it issues system commands to compile and link your application's source files into an executable file. The project Makefile, generated by Project Builder, provides the information **make** needs to do this job. The warnings generated by the compiler and link editor provide information to help you locate and fix bugs detected at compile time.

In building your project, **make** keeps track of source updates. Each time you run **make**, only the source files that have been updated since the last **make** are regenerated; the rest are used as is. This minimizes the time required to generate your executable file.

Once you start building your application, Project Builder provides an interactive interface to Edit for locating source code problems detected by the compiler and link editor. Anytime the compiler encounters an error, Edit can locate the code with a single clickÐyou can then edit out the problem and begin compiling again.

Debug Your Program

After you successfully compile your program, you're ready to try running it. The easiest way to do

so is by choosing Debug in the Project Builder application's Builder display. This selection builds your application (if necessary), then starts GDB in a Terminal shell. You can then run your application with GDB in a couple of ways:

- Use Edit's Gdb panel to step through your application while looking at the code being executed. The Gdb panel provides an easy-to-use, interactive interface that integrates GDB and Edit; it's described in Chapter 2, ªThe Project Builder Application.º

- Run the program from the Terminal shell by issuing GDB commands. The GDB debugger and its commands are described in Chapter 13, ªThe GNU Source-Level Debugger.º

Along with the compiler and GDB, the NEXTSTEP development environment includes several applications and features that can help you trace your program and pinpoint errors. Other developer applicationsÐincluding MallocDebug (Chapter 8), ProcessMonitor (Chapter 9), and Yap (Chapter 10)Ðprovide additional insights into the workings of your program. ProcessMonitor lets you examine various characteristics of any process's activities: memory use, PostScript graphic states, the run-time environment, and so on. MallocDebug measures the dynamic memory use of an application. Yap lets you enter, edit, and execute PostScript code on the fly and allows you to read and write text files so the code can be used elsewhere.

Two tools are available to track off-screen drawing, which may affect what you seeÐor don't seeÐon-screen. The **NXShowPS** argument writes all PostScript code and values from the PostScript interpreter to the standard error stream. The **NXShowAllWindows** argument displays all of an application's windows, including those generated for off-screen imaging. Both of these are command-line arguments. To use them, start your program from a Terminal shell. On the command line, enter the program name followed by the parameter. For example

```
/me/MyApps/NewApp/NewApp.app -NXShowAllWindows
```

starts the application **NewApp.app**, displaying all its windows as it runs.

Add Help to Your Application

Using Project Builder, Interface Builder, and Edit, you can create context sensitive help for your application. The standard help template provided by Interface Builder includes general information on the NEXTSTEP environment. You can add to this template to include application-specific help, and you can create links between the controls in your application and the help system to provide the user with context-specific assistance.

Translate Your User Interface

When the application is complete and help is available, you can create alternate versions with translated text for windows, panels, menu items, and buttons, as well as any help information you've added. NEXTSTEP application programming interface (API) provides ways of accessing bundles in your application containing the text and user interface in various languages you wish to support. ªThe Project Builder Application,º Chapter 2, provides information on how to make a project localizable.

Make Your Application Available to Users

Once an application is debugged, you can install it in an application directory using Project Builder. Project Builder lets you determine which directory to install the application in and provides a way to automatically install the application when you build it.

When the user double-clicks a document file, the Workspace Manager has to locate and start the executable file for that application. Workspace Manager looks for the executable file in a systematic sequence of directory paths. This search sequence is contained in an environmental variable **path**. You can place an application in any of the directories specified in **path**.

Because of the search sequence specified by **path**, you can replace an application located later in the sequence with one of the same name earlier in the sequence. For example, **$(HOME)/Apps** occurs before **/NextApps** in **path**; if you place an application in the directory **$(HOME)/Apps** with the same name as an application in the **/NextApps** directory, the Workspace Manager finds and starts the version in **$(HOME)/Apps** (the Apps subdirectory in your home directory). You should consider the path when naming and installing applications.

If your application is intended for distribution on multiple floppy disks, you should configure it so that a user can install it using the Installer application. Tools for doing so are documented in **/NextLibrary/Documentation/NextDev/Concepts/Installer.rtf**.