Introduction

Object-oriented programming, like most interesting new developments, builds on some old ideas, extends them, and puts them together in novel ways. The result is many-faceted and a clear step forward for the art of programming. An object-oriented approach makes programs more intuitive to design, faster to develop, more amenable to modifications, and easier to understand. It leads not only to new ways of constructing programs, but also to new ways of conceiving the programming task.

Nevertheless, object-oriented programming presents some formidable obstacles to those who would like to understand what it's all about or begin trying it out. It introduces a new way of doing things that may seem strange at first, and it comes with an extensive terminology that can take some getting used to. The terminology will help in the end, but it's not always easy to learn. Moreover, there are as yet few full-fledged object-oriented development environments available to try out. It can be difficult to get started.

That's where this book comes in. It's designed to help you become familiar with object-oriented programming and get over the hurdle its terminology presents. It spells out some of the implications of object-oriented design and tries to give you a flavor of what writing an object-oriented program is really like. It fully documents the Objective-C™ language, an object-oriented programming language based on standard C, and introduces the most extensive

object-oriented development environment currently availableÐOPENSTEP™.

The book is intended for readers who might be interested in:

- Learning about object-oriented programming,
- Finding out about the OPENSTEP development environment, or
- Programming in Objective-C.

NeXT supplies its own compiler for the Objective-C language (a modification of the GNU C compiler) and a run-time system to carry out the dynamic functions of the language. It has tested and made steady improvements to both over the years; this book describes the latest release, which includes provisions for declaring and adopting protocols and setting the scope of instance variables.

Throughout this manual and in other NeXT documentation, the term ªObjective-Cº refers to the language as implemented for the OPENSTEP development environment and presented here.

The Development Environment

Every object-oriented development environment worthy of the name consists of at least three parts:

- A library of objects and software frameworks and kits
- A set of development tools
- An object-oriented programming language

OPENSTEP comes with an extensive library. It includes several software frameworks containing definitions for objects that you can use ªoff the shelfº or adapt to your program's needs. These include the Foundation Framework, the Application Kit™framework (for building a graphical user interface), and others.

OPENSTEP also includes some exceptional development tools for putting together applications. There's Interface

Builder™, a program that lets you design an application graphically and assemble its user interface on-screen, and Project Builder, a project-management program that provides graphical access to the compiler, the debugger, documentation, a program editor, and other tools.

This book is about the third component of the development environmentÐthe programming language. All OPENSTEP software frameworks are written in the Objective-C language. To get the benefit of the frameworks, applications must also use Objective-C. You are not restricted entirely to Objective-C, however; you are free to incorporate C++ code into your applications as well.

Objective-C is implemented as set of extensions to the C language. It's designed to give C a full capability for object-oriented programming, and to do so in a simple and straightforward way. Its additions to C are few and are mostly based on Smalltalk, one of the first object-oriented programming languages.

This book both introduces the object-oriented model that Objective-C is based upon and fully documents the language. It concentrates on the Objective-C extensions to C, not on the C language itself. There are many good books available on C; this manual doesn't attempt to duplicate them.

Because this isn't a book about C, it assumes some prior acquaintance with that language. However, it doesn't have to be an extensive acquaintance. Object-oriented programming in Objective-C is sufficiently different from procedural programming in standard C that you won't be hampered if you're not an experienced C programmer.

Why Objective-C

The Objective-C language was chosen for the OPENSTEP development environment for a variety of reasons. First and foremost, it's an object-oriented language. The kind of functionality that's packaged in the OPENSTEP software frameworks can only be delivered through object-oriented techniques. This manual will explain how the frameworks work and why this is the case.

Second, because Objective-C is an extension of standard ANSI C, existing C programs can be adapted to use the

software frameworks without losing any of the work that went into their original development. Since Objective-C incorporates C, you get all the benefits of C when working within Objective-C. You can choose when to do something in an object-oriented way (define a new class, for example) and when to stick to procedural programming techniques (define a structure and some functions instead of a class).

Moreover, Objective-C is a simple language. Its syntax is small, unambiguous, and easy to learn. Object-oriented programming, with its self-conscious terminology and emphasis on abstract design, often presents a steep learning curve to new recruits. A well-organized language like Objective-C can make becoming a proficient object-oriented programmer that much less difficult. The size of this manual is a testament to the simplicity of Objective-C. It's not a big bookÐand Objective-C is fully documented in just two of its chapters.

Objective-C is the most dynamic of the object-oriented languages based on C. The compiler throws very little away, so a great deal of information is preserved for use at run time. Decisions that otherwise might be made at compile time can be postponed until the program is running. This gives Objective-C programs unusual flexibility and power. For example, Objective-C's dynamism yields two big benefits that are hard to get with other nominally object-oriented languages:

- Objective-C supports an open style of dynamic binding, a style than can accommodate a simple architecture for interactive user interfaces. Messages are not necessarily constrained by either the class of the receiver or the method selector, so a software framework can allow for user choices at run time and permit developers freedom of expression in their design. (Terminology like ªdynamic binding,º ªmessage,º ªclass,º ªreceiver,º and ªselectorº will be explained in due course in this manual.)
- Objective-C's dynamism enables the construction of sophisticated development tools. An interface to the run-time system provides access to information about running applications, so it's possible to develop tools that monitor, intervene, and reveal the underlying structure and activity of Objective-C applications. Interface Builder could not have been developed with a less dynamic language.

How the Manual is Organized

This manual is divided into four chapters and two appendices. The chapters are:

- Chapter 1, ª**Object-Oriented Programming**,º discusses the rationale for object-oriented programming languages and introduces much of the terminology. It develops the ideas behind object-oriented programming techniques. If you're already familiar with object-oriented programming and are interested only in Objective-C, you may want to skip this chapter and go directly to Chapter 2.
- Chapter 2, ª**The Objective-C Language**,º describes the basic concepts and syntax of Objective-C. It covers many of the same topics as Chapter 1, but looks at them from the standpoint of the Objective-C language. It reintroduces the terminology of object-oriented programming, but in the context of Objective-C.
- Chapter 3, ª**Objective-C Extensions**,º concentrates on two of the principal innovations introduced into the language as part of OPENSTEP Objective-CÐcategories and protocols. It also takes up static typing and lesser used aspects of the language.
- Chapter 4, ª**The Run-Time System**,º looks at the NSObject class and how Objective-C programs interact with the run-time system. In particular, it examines the paradigms for allocating and initializing new objects, dynamically loading new classes at run time, and forwarding messages to other objects.

The appendices contain reference material that might be useful for understanding the language. They are:

- Appendix A, ª**Objective-C Language Summary**,º lists and briefly comments on all of the Objective-C extensions to the C language.
- Appendix B, ª**Reference Manual for the Objective-C Language**,º presents, without comment, a formal grammar of the Objective-C extensions to the C language. This reference manual is meant to be read as a companion to the reference manual for C presented in *The C Programming Language* by Brian W. Kernighan and Dennis M. Ritchie, published by Prentice Hall.

Conventions

Where this manual discusses functions, methods, and other programming elements, it makes special use of bold and italic fonts. **Bold** denotes words or characters that are to be taken literally (typed as they appear). *Italic* denotes words that represent something else or can be varied. For example, the syntax

@**interface** *ClassName* **(** *CategoryName* **)**

means that @**interface** and the two parentheses are required, but that you can choose the class name and category name. Where method syntax is shown, the method name is bold, parameters are italic, and other elements (mainly data types) are in regular font. For example:

- (void)**encodeWithCoder:**(NSCoder *)*coder*

Where example code is shown, ellipsis indicates the parts, often substantial parts, that have been omitted:

```
- (void)encodeWithCoder:(NSCoder *)coder
{
    [super encodeWithCoder:coder];
    . . .
}
```

The conventions used in the reference appendix are described in that appendix.