pple
ESS

Includes animated demonstrations of the new Mac OS 8 features

Mac OS 8 REVEALED

A Technical Tour of the New Mac OS

TONY FRANCIS

Library of Congress Cataloging-in-Publication Data
Francis, Tony, 1956–
Mac OS 8 revealed / Tony Francis.
p. cm.
Includes index.
ISBN 0-201-47955-9
1. Mac OS 8. 2. Operating systems (Computers) I. Title.
QA76.76.063F76 1996
005.4'469--dc20 96-25119
CIP

Sponsoring Editor: Keith Wollman
Project Manager: Sarah Weaver
Cover design: Suzanne Heiser
Text design: Kim Arney
Set in 10 point Sabon by Tony Francis and Vicki Hochstedler

1 2 3 4 5 6 7 8 9 - MA - 0099989796
First printing, August 1996

Mac OS 8 Revealed

Tony Francis

Addison-Wesley Developers Press
Reading, Massachusetts • Menlo Park, California • New York
Don Mills, Ontario • Harlow, England • Amsterdam
Bonn • Sydney • Singapore • Tokyo • Madrid • San Juan
Seoul • Milan • Mexico City • Taipei

To Muzzy and Sharon, for teaching me kindness,
and to Jeri Ann, for teaching me the telemark turn.

Contents

List of Figures

Preface

You'll discover a new computing landscape when you encounter Mac OS 8, the next major release of Apple Computer's operating system—formerly code named *Copland*. This guide will acquaint you with the technical geography of Mac OS 8: its architecture, history, and special points of interest.

Who Might Find This Book Helpful

This book offers a technical tour of Mac OS 8 for computer professionals, such as programmers, product managers, engineering managers, systems integrators, information system professionals, and instructional designers. Technically oriented computer enthusiasts might also enjoy this tour.

Developers are individuals or organizations that create software or hardware products for commercial, in-house, or personal use. In this book, the term **users** refers to the people who use the products created by developers.

This book illustrates many Mac OS 8 user features and describes many of the system's user benefits but does so from a developer's perspective—that is, this book describes how these features and benefits might be implemented in software and hardware products. If you're a general computer user, you should look elsewhere for information about Mac OS 8 that might be more relevant to your needs; user-oriented books will become widely available about the time Mac OS 8 begins shipping to customers.

Mac OS 8 Revealed provides an overview of the design and architecture of Mac OS 8 and outlines how these affect the design and architecture of software and hardware products. This book doesn't describe specific programming interfaces for software engineers or show schematic drawings for hardware engineers. But by describing the technical breadth of Mac OS 8, this book will familiarize you with Mac OS 8 so that you can begin exploring its technical details more easily.

How to Navigate This Book

This book is organized to assist readers with different needs and backgrounds.

The Whirlwind Tour

If you have time for only the briefest excursion into Mac OS 8, take a look at the first two chapters. They present a very quick tour of the Mac OS 8 environment and its capabilities.

- Chapter 1 presents highlights of the Mac OS 8 platform from the user's perspective.
- Chapter 2 presents a high-level tour of the Mac OS 8 platform from the developer's perspective.

The remaining chapters of this book elaborate on the material introduced in these two chapters.

Terminology

You might encounter some unfamiliar terminology as you read about Mac OS 8. This book uses several approaches to familiarize you with Mac OS 8 terminology.

Look at the beginning of every chapter for the section "Key Terms and Concepts" to find definitions of principal words and brief explanations of major topics. Here and elsewhere, words appear in **boldface** when first defined. Words appearing in boldface are also listed in the glossary at the end of the book.

Page margins contain definitions of terms that aren't new or unique to Mac OS 8.

Many general computer science terms are left undefined in the main text but are defined in the page margins. Thus, you can easily skip over the definitions of terms with which you are already familiar. Occasionally, definitions presented in earlier chapters are repeated in the margins to refresh your memory.

Side Trips and Shortcuts

This book makes it easy for you to find the fastest route to the information you want. Look for signposts labeled "Mac OS Heritage" and "Compatibility Notes" in the left margins. These signposts mark excursions into information useful to some readers, but not to others.

Look at "Mac OS Heritage" boxes like the one below for discussions of earlier versions of Mac OS technologies.

MAC OS HERITAGE

Look in These Sections for Historical Background Information

If you have little experience with the Mac OS, you'll find sections like these useful. An understanding of Mac OS history illuminates many of the design decisions behind Mac OS 8. These sections also describe much of what hasn't changed in the Mac OS between System 7.5 and Mac OS 8. Experienced Mac developers, however, may want to skip over these historical discussions. Their titles can help you decide whether the information is relevant to you.

If you've already developed a product for the Mac OS, look at "Compatibility Notes" sections like the one below to learn whether or how you need to revise your product to make it run compatibly in Mac OS 8.

COMPATIBILITY NOTES

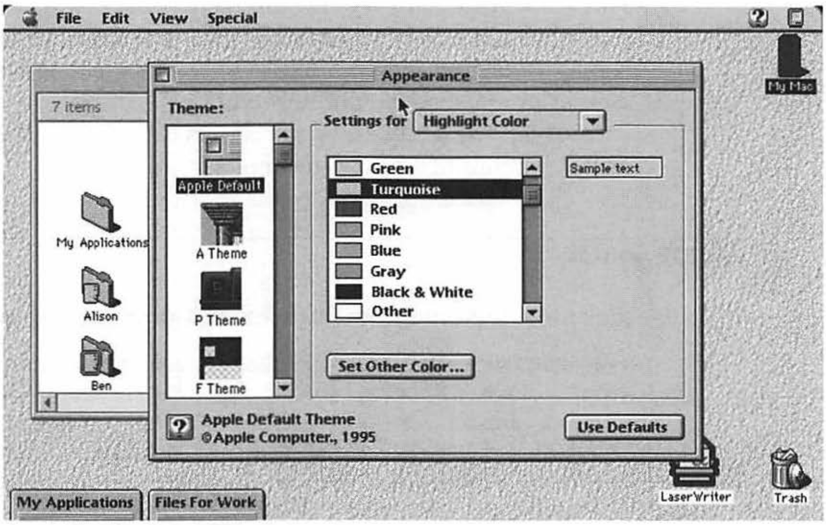
Look Here for Information About Mac OS 8 Support for System 7 Software

Most applications written for System 7 of the Mac OS run compatibly in Mac OS 8 with little or no revision. If you've developed a System 7 product, you'll want to know how and to what degree Mac OS 8 supports software developed for System 7. Sections like these will inform you.

These sections don't describe new Mac OS 8 features and capabilities that developers should consider adopting. So if you're planning to create a new product that takes full advantage of Mac OS 8 features, or if you haven't developed a System 7 product, you'll probably want to skip these sections.

If you're thinking about creating a product for Mac OS 8, examine the chapter-ending sections titled "Planning a Product for Mac OS 8" for development approaches you should consider taking.

FIGURE P.1 A screen shot from Mac OS 8, with an accompanying CD-ROM movie



MAC OS HERITAGE

Mac OS Release Names

The Macintosh operating system preceding Mac OS 8 is called System 7. Since the release of System 7, Apple Computer has embarked on an operating system licensing strategy that emphasizes the brand name "Mac OS" to distinguish the operating system from the company's line of Macintosh computers and the Mac OS–compatible computers available from other manufacturers. For example, Mac OS 8 is supported by Macintosh computers available from Apple Computer and by Common Hardware Reference Platform–compliant computers available from other computer manufacturers such as Motorola and IBM.

HOW TO NAVIGATE THE BOOK CD-ROM

This tour book is intended to introduce you to Mac OS 8's many features so that you can choose where and how to begin to explore them personally. To that end, this book includes dozens of screen shots that illustrate Mac OS 8 capabilities.

For example, the screen shot in the figure above illustrates one of multiple sets of coordinated designs that users can choose to help personalize the appearance of their systems. This figure shows elements from the operating system's default design. Screen shots like these give you picture-postcard glimpses of Mac OS 8 capabilities.

The CD-ROM that accompanies this book provides movie versions of many of these screen shots, thereby offering movie travelogs as well as picture postcards of the Mac OS 8 environment. For example, the CD-ROM movie version of the figure on the preceding page lets you examine elements of several human interface appearances under design at Apple. Both the CD-ROM movie and the screen shot appear onscreen in color if you have a color monitor, giving you a more vivid tour than figures in the book provide.

Movies of Mac OS 8 features are identified by a film strip in the left margins of figures, like that shown on the preceding page. It's easy to find and run these movies on the CD-ROM. You'll need a Mac OS–compatible computer running System 7.5 and the QuickTime system extension. If QuickTime's not currently running on your System 7.5 computer, open the Extensions Manager control panel, turn on the QuickTime extension, and restart your computer.

Then, load the CD-ROM into your drive. If you don't already have Adobe Acrobat Reader 2.1 installed on your system, follow the instructions on the CD-ROM for installing Acrobat Reader 2.1, which is included on the disk. Then open the document titled "Mac OS 8 Revealed on CD." This file is an Acrobat version of the printed book. Every page from the printed book is represented onscreen in the electronic version of the book. To run an animated illustration, simply go to the page in the electronic book that contains a screen shot marked by the film strip and click the mouse button with the cursor anywhere on the figure.

These movies are entirely self-running; you don't need to start up any other software. When a movie finishes, it disappears from your screen, returning you to the page in the book containing the screen shot.

The electronic version of the book is intended to help you locate and start these movies. The text surrounding each movie provides an explanation of—and a context for—the capabilities demonstrated in the CD-ROM movie. However, if you're too impatient to use the electronic book as a navigation tool, you'll also find the movies in the folder titled "Mac OS 8 Demos" on the CD-ROM, where filenames match the figure numbers from the book. You can double-click any of these files to launch a movie.

Also note that directly launching a movie by double-clicking its icon requires only 2 megabytes (MB) of available physical memory, whereas using the CD-ROM version of the book to locate and start movies requires about 5MB. Directly launching movies doesn't require you to install Acrobat Reader 2.1. So if your computer is low on physical memory, or if you don't install Acrobat Reader 2.1, you may still view the demos even without using the electronic version of the book as your navigation tool.

Movie demonstrations of features introduced with Mac OS 8 appear in Chapter 1. For existing System 7 technologies that have been refined and integrated into Mac OS 8 (for instance, QuickTime Conferencing, QuickDraw 3D, and Cyberdog), you'll find movie demonstrations in Chapters 15 and 16.

Cautions About a Changing Environment

Every effort—including close technical reviews by dozens of Apple Computer's key engineers—has been made to ensure that this book accurately describes Mac OS 8 at the moment this book goes into production. However, Mac OS 8 is still under development. Much can and will change between the publication of this book and the release of Mac OS 8 to customers.

At the time this book went to press, Apple Computer was preparing to send early versions of Mac OS 8 and its preliminary technical documentation to thousands of development partners. Apple Computer will almost certainly modify Mac OS 8 in response to feedback it receives from these developers. Even the look and feel of Mac OS 8 as illustrated in this book and CD-ROM is based on various prototype designs that are subject to change.

In other words, you shouldn't depend on Mac OS 8 capabilities shipping exactly as described in this book or appearing exactly as illustrated. On the other hand, Apple Computer is eager to help developers create early Mac OS 8 products that might, in turn, serve to refine the shipping release of the operating system. If you begin development of a Mac OS 8 product, your own efforts might help shape the operating system for the benefit of your customers.

Apple is making prerelease versions of Mac OS 8 available to all members of the Macintosh Associates, Macintosh Associates Plus, and Macintosh Partners developer programs. If you're interested in joining these programs, visit the World Wide Web page at http://dev.info.apple.com for information. The home page for Mac OS 8, located at http://www.macos.apple.com/macos8, contains additional information about the operating system that developers and users may find valuable.

Acknowledgments

I'm grateful to a large number of people at Apple Computer for taking time from their tight schedules to assist me with this book. Enthusiastically explaining Mac OS 8 technology and carefully reviewing drafts of this book, the software engineers were a pleasure to work with. Alan Lillich, Holly Knight, Jeff Cobb, and Russell Williams in particular spent a great deal of time patiently explaining the intricacies of core operating system services. They each offered valuable feedback on drafts of various chapters, and Jeff Cobb graciously examined every chapter. Jeff's comments especially improved the presentation of information from chapter to chapter.

A new operating system is a huge and complex software project, and I'm indebted to a host of other engineers working on portions of Mac OS 8. Barton House and David Harrison tutored me on the operating system's memory architecture. Deeje Cooley and Arno Gourdol teamed up to explain the Assistance Services. Dave Heller helped me with the file system. Chris Linn explained the integration of OpenDoc and Mac OS 8. Tom Dowdy explained the integration of Mac OS 8, QuickDraw GX, and QuickDraw, while Pablo Fernicola helped me with QuickDraw 3D. In addition, these engineers took time to carefully review drafts of various chapters.

I imposed myself on even more engineers, and they also responded kindly. Mark Day offered valuable comments on nearly every chapter. Brian McGhie, Winston Hendrickson, and John Iarocci reviewed various chapters and offered helpful feedback. As the deadline for this book approached, Richard Ford, Glenn Fisher, and Will Stein dropped what they were doing to help me with the last two chapters.

As the technical lead for Mac OS 8, Wayne Meretsky had a tremendous influence on this book. Most saliently, Wayne's architectural design decisions became topics for much of the book. Moreover, Wayne enlisted engineering support for this endeavor since its inception, and he was instrumental in driv-

ing the resolution of a host of nomenclature issues. Wayne also took valuable time out of his hectic schedule to explain topics ranging from operating system concepts to Mac OS 8 implementation details.

Very helpful, too, were the technology evangelists, who offered their insights into how developers might best use Mac OS 8 capabilities. Adam Samuels, Alan Samuel, Bob Selzler, Garry Hornbuckle, Peter Lowe, Rick Carmichael, and Vito Salvaggio each had a hand in improving this book, and I'm grateful for their contributions. Adam and Alan also provided me with the demos that appear on the CD-ROM with this book.

The technical publications staff responsible for producing the *Inside Macintosh* book series shared their talents and gave generously of their time. Antonio Padial did an extraordinary job editing this book, and he has my deep gratitude. I also owe a great deal to Sean Cotter, who provided needed advice, encouragement, and editorial assistance throughout this entire endeavor. I based most of Chapter 12 and Chapter 14 on the excellent seed documentation that Sean has prepared for developers.

In addition, Dee Eduardo wrote superb preliminary documentation that I relied upon in preparing Chapter 11. Her reviews of that chapter, as well her ongoing help with the glossary, were very helpful. Sanborn Hodgkins offered helpful ideas about presenting information in Chapter 10. Greg Williams, Jeanne Woodward, Judy Melanson, Laurel Rezeau, Patria Brown, and Paul Black also offered keen suggestions that dramatically improved the quality of this book.

Lisa Hymel prepared the illustrations. I greatly appreciated her enthusiasm and flexibility as much as her superb work. As always, it was a pleasure to work with Anne Szabla, whose excellent research provided the basis for much of the information in Chapters 15 and 16.

Lorraine Aochi, editor-in-chief of Apple Press, and Ken Bereskin, chief evangelist for Mac OS 8, were champions throughout. Not only did they provide valuable suggestions for improving the book, but Lorraine and Ken also ensured that I received the help necessary to write it. I'm indebted to Lorraine, Ken, and Keith Wollman, my editor at Addison-Wesley, for the support and encouragement that made this book possible.

The largest share of my gratitude goes to Trish Eastman and Sharon Everson, longtime, esteemed colleagues of mine at Apple. Trish, writing manager for *Inside Macintosh*, and Sharon, a lead writer for the series, were generous with assistance, rich with suggestions, tireless in support, and flowing with encouragement. I consider it my great fortune to have had their assistance and collaboration on this project.

1

A Scenic Tour of the User's Environment

This chapter highlights the user experience of the Mac OS 8 platform. In particular, this brief tour offers an early look at Apple Computer's redesigned human interface and a quick peek at its flexible new operating system. Taking a tour from the user's perspective is a valuable way for computer professionals to become familiar with Mac OS 8, as the user experience is usually the starting point for product design.

Many of the advantages supplied by the Mac OS 8 platform—advantages such as its high-performance I/O system, its efficient use of physical memory and CPU time, its system reliability features, and the tight integration of its operating system features—are difficult to represent merely by showing the system's human interface features. However, this chapter introduces many key user features and the key operating system capabilities that support these features.

The screen shots that appear in this chapter help to show Mac OS 8 capabilities, but illustrative movies are supplied on the CD-ROM version of this chapter. If you haven't already done so, read "How to Navigate the Book CD-ROM" in the preface for information about using the CD-ROM version of this book, then load the CD-ROM on your Mac OS–compatible computer and join the tour.

MAJOR POINTS OF INTEREST

Mac OS 8 offers a wealth of features and capabilities to users and developers. This chapter presents just a few of the operating system's most visible innovations—in particular, this chapter shows how Mac OS 8 supplies users with a flexible, easy-to-use, and productive computer environment.

As you'll see in this chapter, Mac OS 8 allows users to scale its features flexibly according to differing needs and experience levels, even when several users share the same computer. After choosing environments offering desired feature sets, users can further personalize the way their computers look.

By adhering to the Common Hardware Reference Platform (CHRP) architecture, the operating system also gives users the flexibility of choosing the right computer for their needs from models produced by various computer manufacturers.

Mac OS 8 provides new features that significantly improve a human interface historically noted for its ease of use. For example, small programs known as **experts** perform as much automation as possible while assisting users in performing otherwise complex or difficult-to-remember tasks. The operating system even recognizes when users repeatedly perform operations inefficiently; the system then informs users of useful shortcuts. Mac OS 8 also makes it much easier for users to organize and find information on their own systems and on network systems through new capabilities for directly manipulating onscreen information and through relational searches of document contents.

Mac OS 8 attains new levels of raw computing performance through its optimized use of the PowerPC microprocessor architecture, its preemptively multitasked scheduling of CPU use, its efficient use of physical memory, and its fully reentrant I/O system. These capabilities in turn increase the performance of traditional Macintosh strengths in such areas as graphics, multimedia, and networking and communications. And so that users can continue using their favorite System 7 applications, Mac OS 8 provides backward application compatibility.

There's another point you might discover by viewing the demonstration movies on the CD-ROM: Mac OS 8, like previous versions of the Mac OS, makes computers more fun to use, too.

FLEXIBILITY IN THE COMPUTER ENVIRONMENT

To help accommodate the widely ranging and ever-changing demands that users make of their computers, Apple has built a significant degree of flexibility into Mac OS 8. For example, users can

- scale Mac OS 8 features according to differing needs and experience levels
- personalize the Mac OS 8 human interface to better suit individual tastes
- run Mac OS 8 on a wide variety of industry-standard, Common Hardware Reference Platform computers available from multiple manufacturers
- rely on one operating system that supports their native writing systems
- continue running their existing System 7 applications even as they begin adopting applications that incorporate new Mac OS 8 capabilities

A Scalable User Interface

New features alone don't make a computer more useful. Instead, by their sheer number and complexity, more features often overwhelm users. However, as users become more proficient with—and more demanding of—their computers, they require a system that is as feature-rich as possible.

Mac OS 8 accommodates a wide range of users, from novice to expert, by offering a scalable user interface that allows users to set its degree of complexity. For example, a beginning computer user can choose a scaled-down workspace consisting of a few simple features and a small number of applications, while an advanced user can choose a more complex workspace with all of the system's features and with many highly specialized applications.

Users can set Mac OS 8 to present only basic commands, keeping the computing workspace as simple as possible. Figure 1.1 illustrates a workspace appropriate for a beginning user. The user launches applications by using the mouse to click simple buttons onscreen. If the user is a child, an adult can limit access to important data. Notice that the Trash icon is missing from Figure 1.1—a novice user cannot inadvertently delete files from this system.

Intermediate users can choose to add features to their workspaces, as in Figure 1.2. In this workspace, the user can open and remove folders and files that are inaccessible in the novice workspace pictured in Figure 1.1. More complex system configuration commands are available to the intermediate user as well. For example, the intermediate user can change various system settings.

FIGURE 1.1 A simple workspace for a novice user

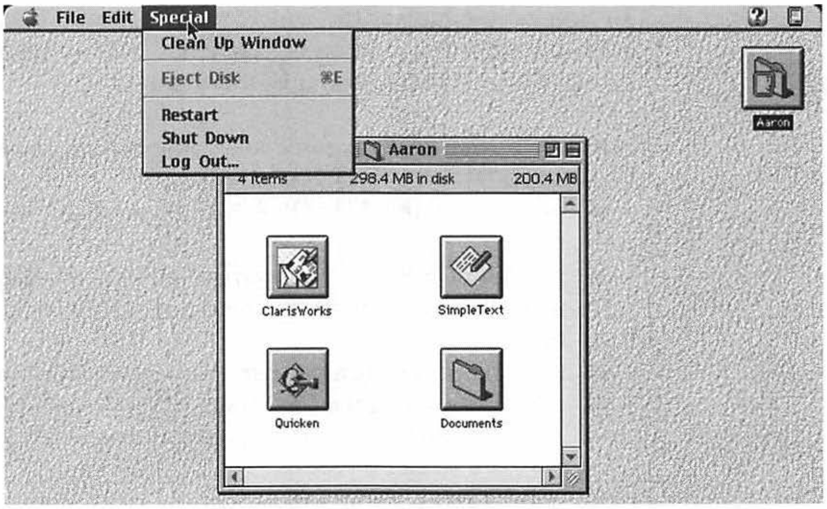


FIGURE 1.2 A workspace for an intermediate user

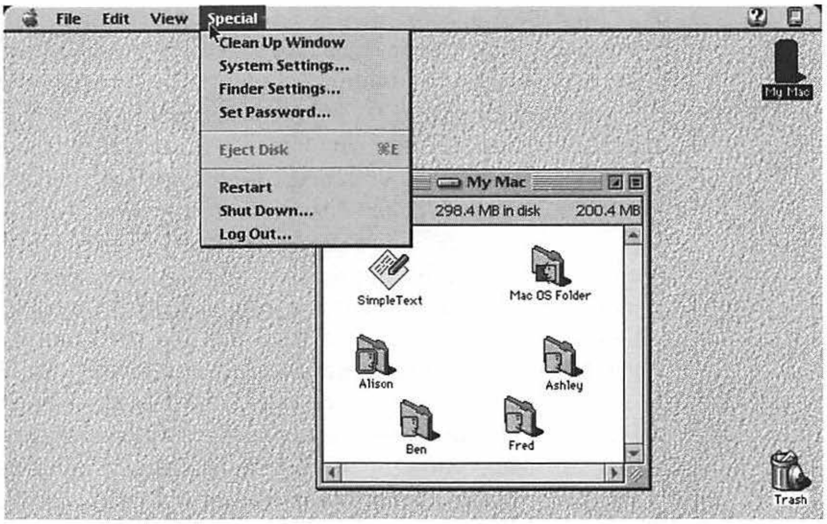


FIGURE 1.3 A workspace for an advanced user

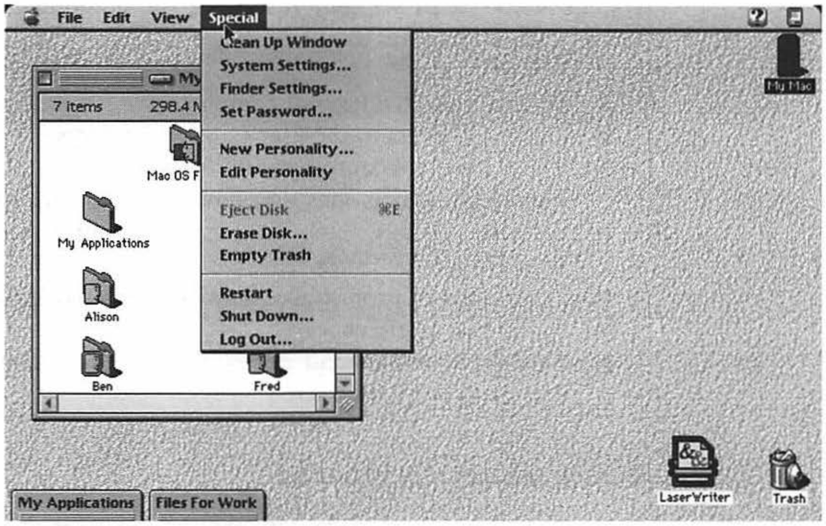


Figure 1.3 shows a workspace for an advanced user. An advanced user can enable all system commands and take advantage of the full range of power available with advanced Mac OS 8 features.

The CD-ROM animated version of Figure 1.3 demonstrates how users can set up their Mac OS 8 workspace according to their needs and skill levels. The experienced user has access to all features of the system. Many of these features are filtered out of the workspace for a novice user who shares that system, thereby preventing the novice user from being distracted or overwhelmed by an undesired level of complexity. Software developers can add scalability to their own products, too. For example, a page-layout program might offer different sets of features appropriate to various skill levels—a novice user might see only a basic set of commands and page-layout templates for creating daily restaurant menus or simple school newsletters, whereas an advanced user might see a full range of commands appropriate for large-scale, professional publishing jobs. This flexibility is designed to help beginning users become competent at a basic skill level before learning intermediate features and then progressing comfortably to more advanced features.

The CD-ROM movie version of Figure 1.3 also illustrates how several users can share a computer, with each user maintaining a distinct workspace on the computer. A large proportion of Mac OS–compatible computers are shared by multiple users. Adults might use a home computer to manage finances, catch up on work from the office, and send e-mail, whereas children might use the same computer for homework and play. In a business setting, one high-end Mac workstation might serve as a video production system for an entire work group. User workspaces allow several people sharing a computer to configure it to suit individual tastes, needs, and skill levels. Users can change workspaces without restarting the computer. A workspace encompasses such preferences as

- the user's desired level of complexity
- access by others to the user's files and folders
- which programs automatically start up when the user begins using the system and which ones shut down when the user finishes
- network settings, such as e-mail account names and bookmarks to World Wide Web sites
- a password to protect access to the user's workspace
- application preferences
- the overall design and appearance of the Mac OS 8 environment, as personalized by the user

Personalized Workspaces

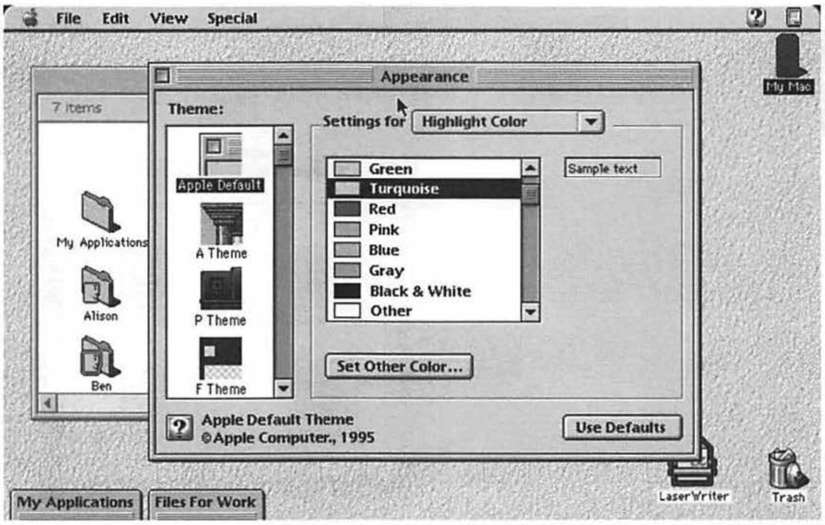
After choosing a workspace with the desired level of complexity, a user can personalize the way the system looks and sounds. Mac OS 8 supplies multiple sets of coordinated designs that affect the appearance of windows, menus, fonts, controls, and other onscreen elements. From among these design sets, called **themes**, a user can choose one that suits the person's mood and taste. Even after choosing a theme, the user can further personalize it by altering such details as the background pattern (also known as the desktop pattern), the screen saver and other desktop animations, the highlight color for selected text, and the system font appearing in menus, dialog boxes, window titles, and the like.

Figure 1.4 illustrates the default theme permanently built into Mac OS 8. This theme is an update of the traditional Mac OS appearance. Users select settings in the Appearance control panel, shown in this figure, to choose themes and to customize the details within a theme.

Mac OS 8 supplies as standard several additional themes as well. (For testing, support, and documentation purposes, developers can rely on the default theme, illustrated in Figure 1.4, to be available on every system.) Figure 1.5 shows how the Appearance control panel and the system look after the user chooses a different theme. Theme changes take effect immediately; the user doesn't have to restart the computer.

Themes can reflect any number of moods. Figure 1.6 shows a whimsical design for children. Each particular theme incorporates matching textures, colors, sounds, and three-dimensional effects. In future releases of Mac OS 8, developers will be able to create additional themes for users to install, keeping the user's interaction with the system fresh and interesting.

FIGURE 1.4 The default theme



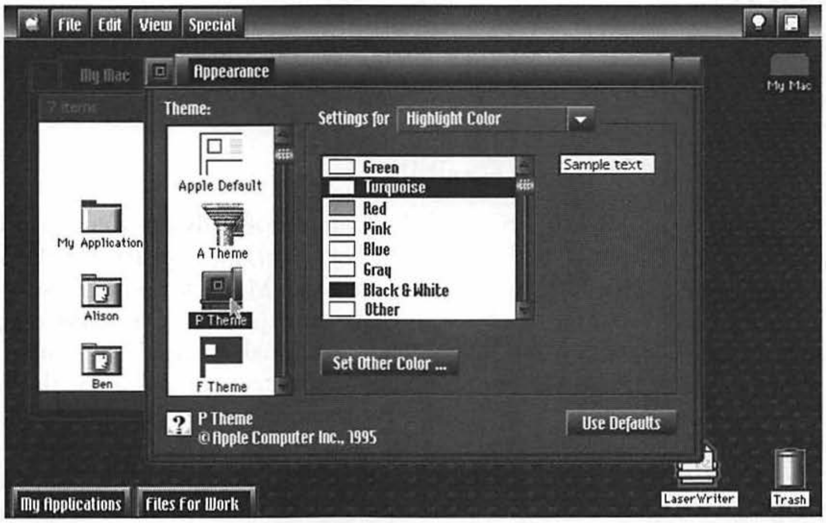
Settings for: Highlight Color

Green
Turquoise
Red
Pink
Blue
Gray
Black & White
Other

Sample text

Apple Default Theme
©Apple Computer., 1995

FIGURE 1.5 A futuristic-looking theme



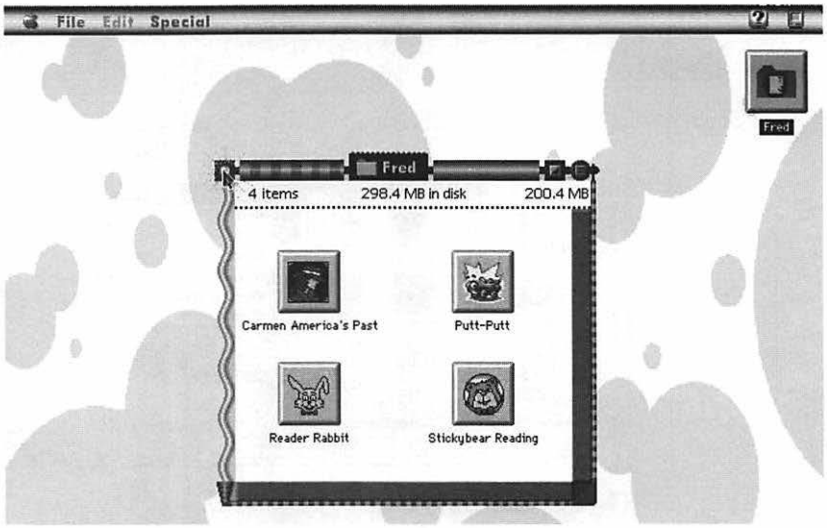
Settings for: Highlight Color

Green
Turquoise
Red
Pink
Blue
Gray
Black & White
Other

Sample text

P Theme
© Apple Computer Inc., 1995

FIGURE 1.6 A theme for children



The CD-ROM animated version of Figure 1.6 also illustrates how Mac OS 8 incorporates themes into the human interface elements of applications, thereby coordinating the appearance of all applications across the system, no matter what theme the user has chosen. Developers don't need to bother with the details of these themes. Instead, developers need deal only with abstractions of the system appearance. At execution time, Mac OS 8 draws the elements constituting a theme.

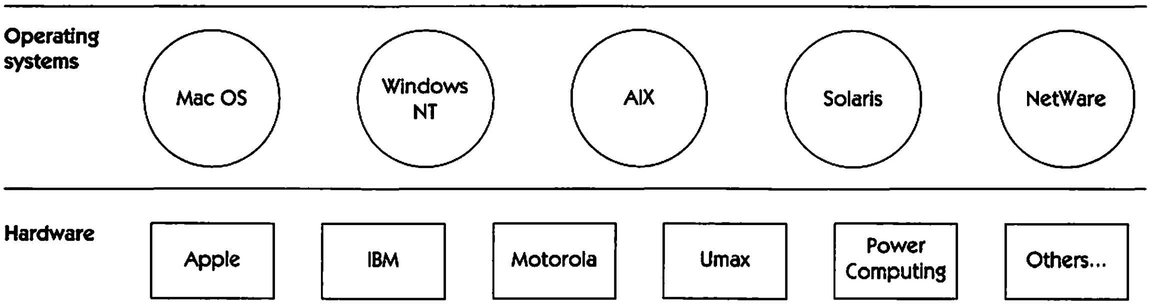
For information about how the operating system implements themes, see **Chapter 12**.

Hardware Flexibility

Mac OS 8 runs on the Common Hardware Reference Platform (CHRP, also known as the PowerPC Platform), an industry standard architecture that any manufacturer can use to build Mac OS–compatible computers. Until 1995, customers who wanted to run the Mac OS could purchase computers only from Apple Computer. Although several computer manufacturers have already begun producing computers compatible with the Mac OS, the CHRP makes it much easier for them and other manufacturers to produce Mac OS–compatible computers.

CHRP-compliant computers can also run 32-bit operating systems from vendors other than Apple Computer. Figure 1.7 illustrates some of the computer manufacturers who intend to produce CHRP-compliant computers, and several of the operating systems being developed to run on this platform. This wide range of choice gives computer buyers the flexibility to purchase various systems according to their needs. Within a company, for example, the graphics

FIGURE 1.7 Common Hardware Reference Platform (CHRP) flexibility



design department can use IBM computers to run applications written for Mac OS 8 while the human resources department can use Apple computers to run applications written for IBM's AIX operating system.

Because various vendors make CHRP-compliant computers, users have a wide choice about the most appropriate computer for running Mac OS 8. For example, a computer from one manufacturer might have features well suited for a school environment, whereas a model from another manufacturer might have the features and device support suitable for a video production environment. By specifying standards for many common computer components, CHRP helps to reduce the cost to purchasers of either system.

CHRP offers a hardware vendor the opportunity for a higher return on investment. By developing a single system that runs multiple operating systems and, hence, a greater number of applications, the vendor can reach a greater number of markets. CHRP provides specifications for input/output (I/O) interfaces, bus standards, and other system-level functional elements.

A **bus** is a path along which information is transmitted electronically within a computer. Buses connect computer devices, such as processors, expansion cards, memory, and peripheral devices.

The modular design of the Mac OS 8 I/O system, in turn, simplifies the work necessary for vendors to extend Mac OS 8 and differentiate their own value-added hardware systems. For instance, vendors can incorporate into their products distinguishing peripheral devices such as laboratory equipment, multimedia appliances, or devices to assist users with disabilities.

Worldwide Text Support

Mac OS 8 supports the features of all of the world's modern writing systems, including those using vertical writing, such as Chinese, those reading from right to left, such as Arabic, and complex contextual languages, such as the writing systems of India. Mac OS 8 allows users who communicate in several languages to mix and print the different writing systems of these languages within a single document—for instance, a multilingual warranty or owner's guide.

Apple Computer has localized the Mac OS for more than 30 geographic markets in over 140 countries, and international text support is further strengthened in Mac OS 8. For instance, Mac OS 8 supports Unicode—an international text-encoding system that provides a code for every character in every major writing system. This encoding standard improves cross-platform compatibility, making it easier for users to collaborate internationally with written documents. Because of the open font architecture of Mac OS 8, font formats for multiple languages can coexist easily on a user's system. Mac OS 8 also enhances support for market-specific features—for instance, speech support for Asian-language text entry.

Chapter 12 discusses the operating system's international text-handling features.

From a developer's perspective, this worldwide text support simplifies the creation of software products for the global market. A developer can prepare a single code base for a product and then localize it for specific national markets. For example, the developer can translate the text displayed by the software to the language and writing system of the target market, translate the user guides, and otherwise change the product to make it culturally acceptable to the target market. A program that allows users to create and manage pages on the World Wide Web, for example, could be prepared for release into dozens of major world markets. By adopting a single, globalized product like this, a multinational organization with web sites originating in several countries lowers its support costs and reduces its internal development efforts.

System 7 Application Compatibility

Mac OS 8 also gives users the flexibility of running most of their existing System 7 applications simultaneously with newer Mac OS 8 programs. This flexibility helps users preserve their existing software investments while incrementally adopting software that takes fuller advantage of the new features and capabilities of the operating system. However, other types of System 7 software—namely, system extensions, control panels, desk accessories, and hardware device drivers—are incompatible with Mac OS 8. Mac OS 8 provides the facilities for developers to create higher-performing and more reliable alternatives to these types of software.

A **system extension** is code that's loaded into memory at system startup time. A **control panel** lets users specify basic settings and preferences for a systemwide feature, such as the speaker volume. A **desk accessory** is a utility available from the Apple menu. A **device driver** is code that controls a device, such as a disk drive.

Since the beginning of Mac OS 8 development, a major priority for Apple Computer has been compatibility with applications and OpenDoc part editors conforming to System 7 development guidelines. The vast majority of these can run without modification in Mac OS 8. However, another major priority for Mac OS 8 is system stability. To improve system stability, Mac OS 8 doesn't support software products that alter or extend the behavior of System 7 or System 7 applications. Developers know these software products as extensions, control panels, and desk accessories. Users find these products highly useful when they present no conflicts with other programs, but maddening when conflicts lead to system unreliability. As alternatives to this type of software, Mac OS 8 supplies developers with far more reliable mechanisms for extending users' systems.

Chapter 9 describes the new software extensibility mechanisms that Mac OS 8 provides to developers. **Chapter 11** describes the I/O architecture.

Improved performance and flexibility are important design goals in the area of device I/O. To accommodate these goals, Apple Computer has designed a higher-performing I/O system. Most System 7 drivers, such as those controlling hardware devices, are incompatible with this new I/O system. However, the design of the Mac OS 8 I/O system makes it much simpler for hardware vendors to write or adapt device drivers for their products.

AN EASIER-TO-USE PLATFORM

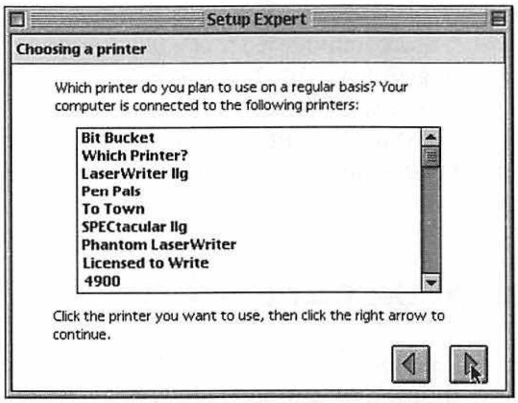
Since the introduction of the Macintosh computer, ease of use has been its greatest strength. Even as computers have become more complex and applications more powerful and intricate, each new version of the Mac OS human interface has become easier to use. You've already seen how this interface in Mac OS 8 can be scaled to levels of complexity appropriate for its users. Users of all skill levels also benefit from the assistance and information navigation features introduced with Mac OS 8. With the assistance features, applications can supply a simple interface allowing users to delegate operations to their computers, and programs can even undertake complex operations on behalf of users. Mac OS 8 also actively offers tips for using the computer more productively.

The Computer as Assistant

Developers can design small Mac OS 8 applications that automate as much of the user's work as possible. An application focused on automating computer operations to help the user is called an **expert**. An expert interviews users so that it can help them carry out complex or seldom-used operations. For example, a setup expert provided by Apple Computer consults the user about configuring a computer, as illustrated in Figure 1.8. The CD-ROM animated version of this figure demonstrates how the expert automatically determines important details on behalf of the user, such as whether the user's computer is connected to a network and, if so, what network services are available. The setup expert then gathers important information that the expert itself can't determine, such as which network printer to select for use. Even when presenting the user with printer selection options, the expert makes various useful assumptions for the user. If the user's computer is connected to an AppleTalk network, for example, the expert displays printers located in the user's local network zone.

Avoiding technical language, the setup expert asks general questions about the user's goals. For example, instead of asking whether the user wants to turn

FIGURE 1.8 A setup expert helping the user configure a computer



Which printer do you plan to use on a regular basis? Your computer is connected to the following printers:

Bit Bucket
Which Printer?
LaserWriter IIg
Pen Pals
To Town
SPECtacular IIg
Phantom LaserWriter
Licensed to Write
4900

Click the printer you want to use, then click the right arrow to continue.

on file sharing or enable guest access, the setup expert asks, "Would you like to share files over the network?" Then it begins to help the user configure file sharing accordingly.

For operations that can be scheduled for later execution, Mac OS 8 supports delegation. Delegation means that the user needn't be at the computer to be productive. For example, a user might wish to have an e-mail application dial an online service and automatically check for mail every hour. While the user is away from the computer, the application still checks for mail. Such delegated operations can also be performed in the background. While a user is running a page-layout program, for example, a program can concurrently dial the online service and alert the user whenever new mail arrives. Only when new mail arrives and the user is ready to read it would the user need to open the e-mail application.

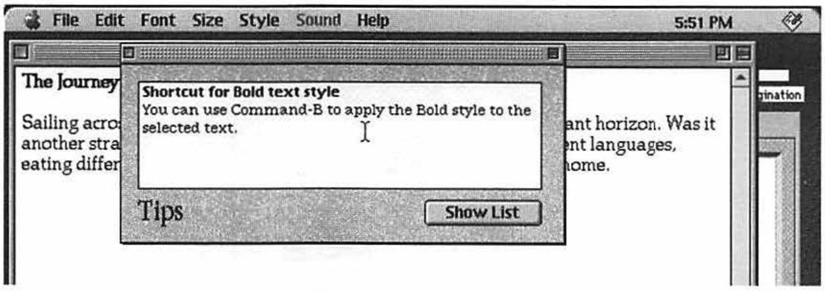
Assistance is also offered to the user by way of **tips,** which are instructions for making more efficient use of application features. In Mac OS 8, tips can appear automatically exactly when a user can benefit from learning them. Part of every workspace includes preferences for the invocation and display of user tips. For example, suppose a user who wants to learn about tips repeatedly applies the boldface character format to text by selecting a command from the Style menu. Mac OS 8 detects this as an inefficient technique and displays a suggestion for using a keyboard shortcut to apply the boldface style, as shown in Figure 1.9.

See **Chapter 13** for information about how developers incorporate these assistance features into their programs.

Information Navigation

Because of the wealth of information that computers and computer networks make available to users, the task of organizing and collecting information has

FIGURE 1.9 A tip for using an application more efficiently



Shortcut for Bold text style
You can use Command-B to apply the Bold style to the selected text.

become more complex. Mac OS 8 simplifies information access and management for users in many ways, such as by offering methods for

- navigating folders
- searching for content
- accessing document information
- organizing information on the desktop

Folder Navigation

The Macintosh Operating System pioneered direct onscreen manipulation of information, and Mac OS 8 improves this capability. For example, Mac OS 8 makes it easier to move files between disks and folders. In System 7, to move a document to a location nested in several levels of folders, the user typically opens one folder after another to get to the target folder. The user drags the document into the window for the target folder and then closes the windows for folders opened along the way.

Folder navigation takes far fewer steps in Mac OS 8. As shown in Figure 1.10, the user drags a document over a disk or folder icon and pauses. In response, a window opens for that disk or folder, as shown in Figure 1.11. By dragging and pausing over additional folder icons, the user navigates to the target folder. Once the user drops the document onto its destination, Mac OS 8 leaves the window for the target folder open but closes other windows opened along the way.

Content Searches

Computer users face the task of organizing and finding information from a variety of sources. The Macintosh Operating System has long supported the hierarchical organization of files and folders, but Mac OS 8 now allows the user to organize information relationally—that is, by grouping items according to user-specified criteria.

FIGURE 1.10 Dragging a file onto a folder icon

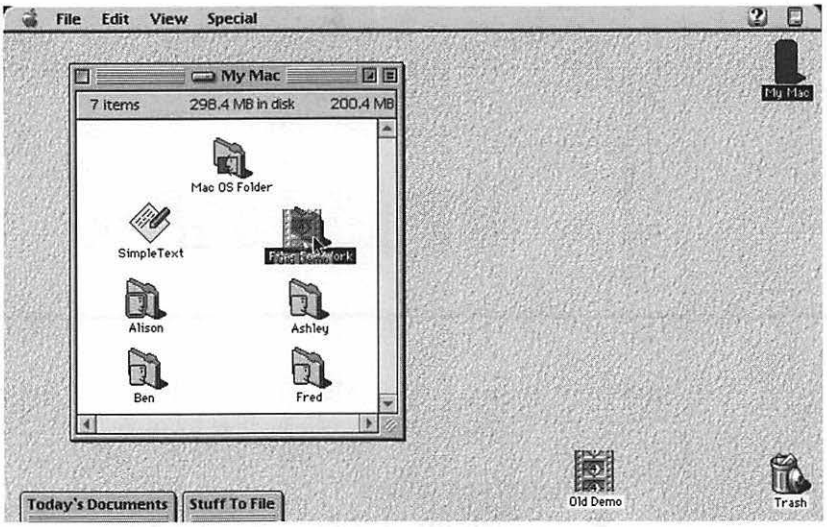


FIGURE 1.11 Navigating folders by dragging a file to another icon

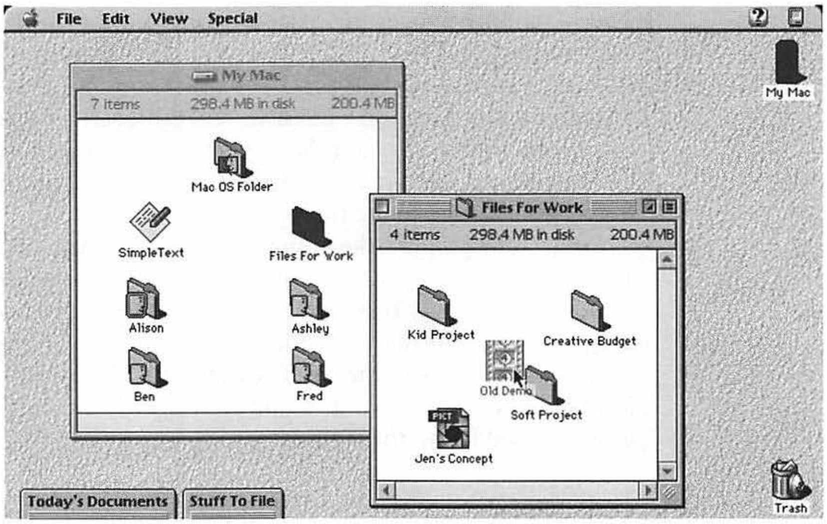
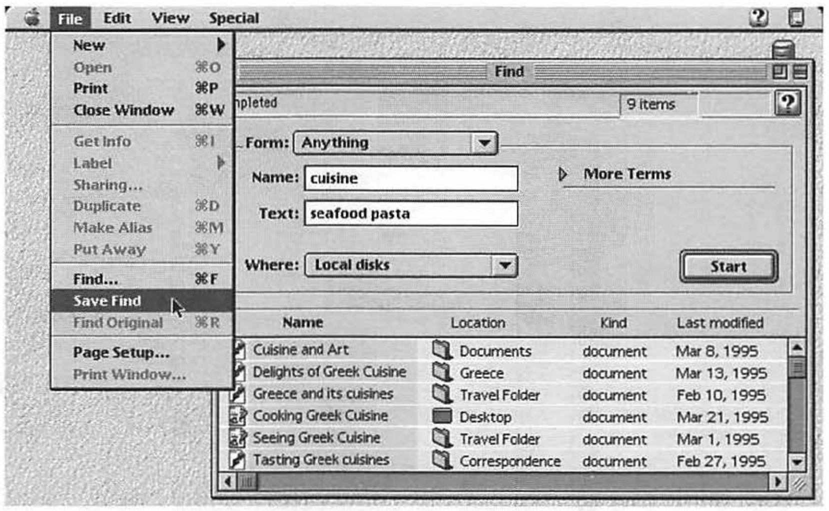


Figure 1.12 Saving the contents of a Find window



pleted 9 items

Form: Anything

Name: cuisine

Text: seafood pasta

Where: Local disks

Name	Location	Kind	Last modified
Cuisine and Art	Documents	document	Mar 8, 1995
Delights of Greek Cuisine	Greece	document	Mar 13, 1995
Greece and its cuisines	Travel Folder	document	Feb 10, 1995
Cooking Greek Cuisine	Desktop	document	Mar 21, 1995
Seeing Greek Cuisine	Travel Folder	document	Mar 1, 1995
Tasting Greek cuisines	Correspondence	document	Feb 27, 1995

Mac OS 8 provides searching and organizing capabilities for any type of information located on any connected storage device containing Mac OS files—including files on network servers. Figure 1.12 illustrates the results of a search for documents whose titles include the word "cuisine" and whose contents include the text "seafood pasta." Mac OS 8 performs the search in the background so that the user can continue working until the search is completed. Meanwhile, the results of the search are continually updated in a Find window.

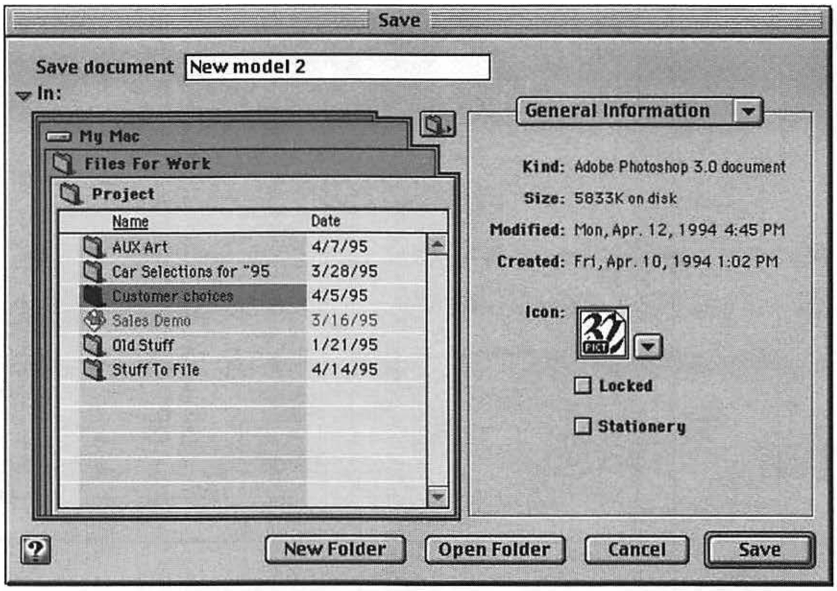
The user can directly act on the items listed in the Find window. After locating a document, for example, the user can rename the document or drag it to a different folder. The user can even save Find windows as special kinds of folders that appear as icons, and the user can store them anywhere on the system. The contents of these folders are kept up to date automatically in the background. For example, if the user logs into a file server that contains documents matching the search criteria saved with a Find window, this window or its folder is automatically updated to include the new documents.

Access to Information About Documents and Folders

See **Chapter 10** for a description of document-navigation services.

Mac OS 8 improves the process of opening and saving documents by giving users better access to documents and to document-related information. Figure 1.13 illustrates a dialog box for saving a file. A panel on the left side of the dialog box displays the contents of a folder. The names of parent folders are also visible, helping the user to navigate the hierarchy of nested folders.

FIGURE 1.13 Document-related information for opening files



Save document: New model 2

Name	Date
AUX Art	4/7/95
Car Selections for "95	3/28/95
Customer choices	4/5/95
Sales Demo	3/16/95
Old Stuff	1/21/95
Stuff To File	4/14/95

Kind:	Adobe Photoshop 3.0 document
Size:	5833K on disk
Modified:	Mon, Apr. 12, 1994 4:45 PM
Created:	Fri, Apr. 10, 1994 1:02 PM
Icon:	

☐ Locked

☐ Stationery

From multiple panels available on the right side of this dialog box, the user can learn a great deal about the items in a folder. For example, the user might see a small representation of the contents of a document, determine its file format, and see what comments might be saved with it.

Organizing Information on the Desktop

Mac OS 8 employs a new type of window called a **pop-up window** that lets users store and quickly access their often-used documents and software programs without adding clutter to their screens. When closed, pop-up windows are identified by title bars that appear on the bottom of the screen, as shown in Figure 1.14. When the user clicks a title bar, the pop-up window opens to display its contents, as illustrated in Figure 1.15. The user can, as with any window, move items in and out of a pop-up window. When a user clicks the title bar of an open pop-up window, the window collapses again and moves to the bottom of the screen. Alternatively, when the user drags an icon to a pop-up window title, the window automatically opens. When the user drops the item in the window, it collapses again.

FIGURE 1.14 Pop-up window title bars

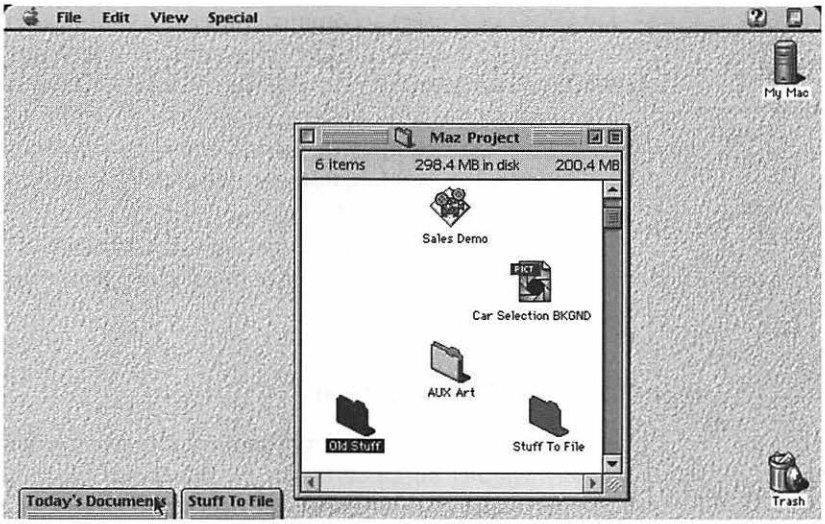
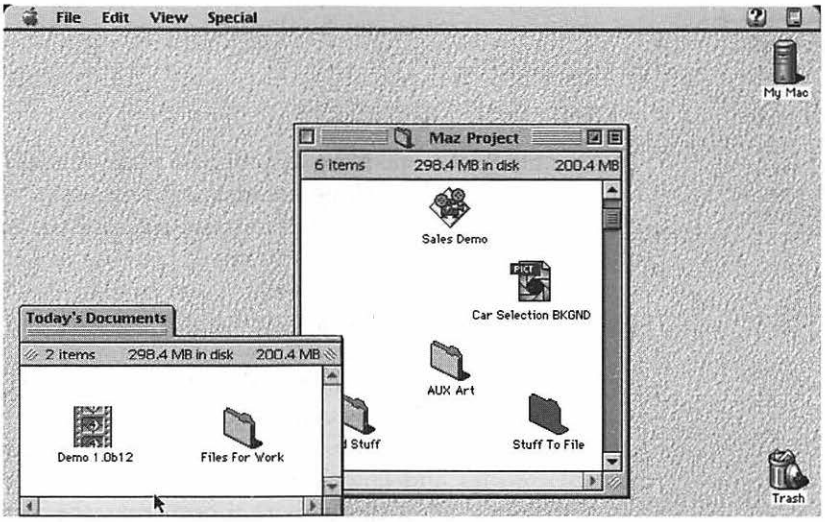


FIGURE 1.15 An opened pop-up window



INCREASED SYSTEM PERFORMANCE

Mac OS 8 helps users accomplish more in less time not only by making applications easier to use, but also by taking greater advantage of the PowerPC microprocessor's performance capabilities. Virtually all of the code for Mac OS 8 is optimized for the RISC architecture of the PowerPC CPU, and Mac OS 8 uses preemptive multitasking to keep the CPU as busy as possible. In this way, Mac OS 8 achieves higher levels of performance than System 7 attains even using the same PowerPC CPUs.

The **CPU** (central processing unit) is the microprocessor that executes instructions and transfers information to and from other high-speed devices (such as physical memory). **RISC** (reduced instruction set computing) is a CPU design featuring the rapid execution of simple machine instructions.

By capitalizing on these advantages, the Mac OS 8 file system—indeed, the entire I/O system—also provides significant performance improvements over System 7.

PowerPC Optimization

Users of PowerPC-based computers are already familiar with the high processing performance supplied by their RISC-based CPUs. Mac OS 8 users will see additional performance increases over earlier versions of the Mac OS because Mac OS 8 operating system features are implemented in PowerPC **native code**—that is, code generated to take fuller advantage of the PowerPC processor architecture. (The performance of many parts of System 7.5 is somewhat hampered because, even on PowerPC-based computers, these portions operate in a mode that emulates Motorola's older 68K CPUs.) For example, users will find that Mac OS 8 documents open faster, windows update more quickly, and network transactions finish sooner.

See **Chapter 8** for more information about how the run-time environment takes advantage of the PowerPC CPU while supporting applications generated for the 68K CPU.

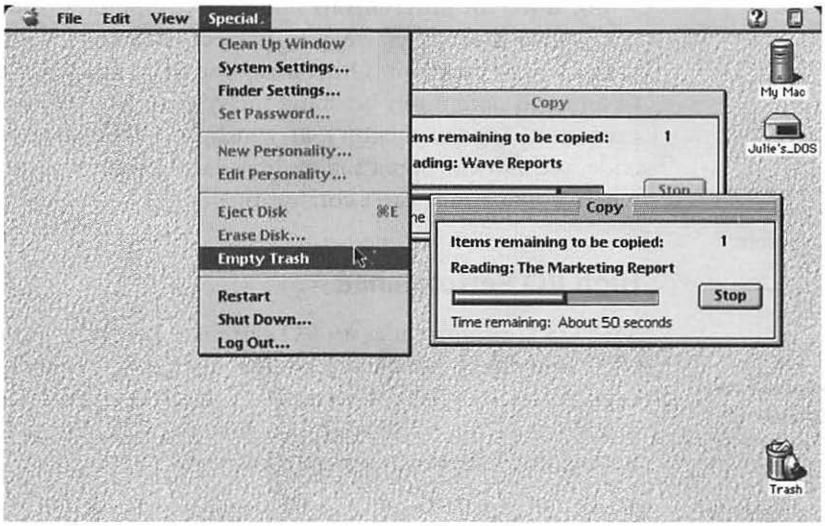
Multitasked Operations

In Mac OS 8, the operating system and multiple software programs are able to run concurrently, making the most efficient use of computer resources. For example, Mac OS 8 might temporarily suspend the execution of a program that is copying a file across a network in order to let a multimedia program prepare video data for immediate onscreen play. At the next opportune CPU cycle, Mac OS 8 resumes execution of the network file copy operation.

Figure 1.16 illustrates multiple tasks operating concurrently. The user has begun copying one large file to a disk drive connected on a network and a second large file to a directly connected drive. As the CD-ROM animated version of this figure demonstrates, before either copy operation is finished, the user can delete files and launch and interact with multiple applications. At the same time that all of this activity is occurring, the computer can also execute other operations that are invisible to the user—for instance, it can send and receive e-mail, serve World Wide Web pages to remote users, and compress or decompress large files. This interleaving of concurrent computer operations allows users to work more productively. A user can continue entering infor-

See **Chapter 4** for details about the multitasking environment.

FIGURE 1.16 Performing multiple operations concurrently



Copy

ms remaining to be copied: 1

ading: Wave Reports

Copy

Items remaining to be copied: 1

Reading: The Marketing Report

Time remaining: About 50 seconds

mation into a spreadsheet even while the computer executes spreadsheet calculations and other tasks in the background, for example.

The operating system interleaves the execution of multitasked operations so quickly that it looks as if they're happening simultaneously on a computer with one CPU. Mac OS 8 also supports multiprocessing so that multiple tasks actually *do* run simultaneously on a computer with more than one processor.

Chapter 5 provides more information about multithreaded programs.

A developer can increase program efficiency by dividing operations into separate, concurrent threads of execution. For example, one thread of execution in a program might handle user interactions, another might perform calculations, and a third might perform I/O. The operating system manages these threads so that they take the greatest advantage of computer resources. For example, while the I/O thread of a program waits for data to come in from a disk and the user interaction thread waits for user input, the calculation thread can make use of the CPU, which might otherwise be idle.

Memory Efficiency and Protection

See **Chapter 6** for a discussion of the virtual memory system and **Chapter 8** for details about the use of shared libraries.

Running multiple programs concurrently requires more memory than running only one program by itself. To support the concurrent operation of multiple programs, Mac OS 8 uses secondary storage, such as a hard disk, to efficiently extend physical memory, allowing the user to open many large applications at one time. Mac OS 8 also makes efficient use of memory by implementing operating system code in shared libraries so that no more than one copy of a shared library is ever needed in memory for use by all programs opened by the user.

For different programs to share the computer at the same time, the operating system must offer ways to protect the memory containing the code and data of each program. Otherwise, one program could corrupt the operations of another, potentially crashing the system. Mac OS 8 offers various forms of protection for the memory in which code and data reside; for example, all code and critical operating system data reside in protected areas of memory where applications can't corrupt them.

Chapter 3 describes the memory-protection facilities.

High I/O Performance

Mac OS 8 also provides an I/O system that's fully optimized for the PowerPC CPU and takes advantage of the operating system's efficient multitasking capabilities. For example, multiple file I/O operations can be concurrently processed in the background so that the CPU doesn't waste valuable cycles waiting for a file request to be executed.

Look in **Chapter 10** for a discussion of the file system. **Chapter 11** describes the general architecture of the entire I/O system.

Higher-Performing Versions of Macintosh Technologies

System 7 offers technologies that have helped make Macintosh users the most loyal customers of any computing platform. Many of these technologies are popular because they allow users to express ideas through graphics, text, animation, video, sound, and network collaboration. In Mac OS 8, these technological strengths are refined and integrated more completely with each other and with the operating system.

For a more thorough description of these landmark Macintosh technologies and how they fit into Mac OS 8, see **Chapter 15** and **Chapter 16**. The CD-ROM versions of these chapters also offer animated demonstrations of several of these technologies.

The use of shared libraries throughout the system helps decrease memory requirements for these technologies. In System 7 users have to install the code for various graphics, multimedia, and communications capabilities as optional system extensions. For example, code for the QuickDraw GX graphics system, the ColorSync color matching system, and the QuickTime multimedia system are loaded when the computer starts up and occupy memory whether or not a user opens an application that uses them. In Mac OS 8, each of these technologies is placed into physical memory only when an application actually uses that technology.

The multitasking capabilities of Mac OS 8 in combination with its I/O system offer more efficient data throughput to increase the performance of all products, but this benefit can be most easily seen in the performance of multimedia products. A video-editing program can read data from a CD-ROM disk in one thread of execution, write data to a hard disk in another thread of execution, and yet remain highly responsive to the user in a third thread. Even when data is slow in coming from the CD-ROM or going to the hard disk, the program can keep the CPU busy executing other operations on behalf of the user.

The efficient memory management of the operating system allows the user to work with many more large applications than can fit in the physical memory of a computer running Mac OS 8 or even the virtual memory of a computer running System 7. This memory efficiency allows the user to work simultaneously with more applications that take advantage of these technologies, such as memory-hungry graphics and multimedia applications.

Chapter 6 describes the virtual memory system.

Graphics and Multimedia

The QuickDraw, QuickDraw GX, and QuickDraw 3D graphics systems are integrated into Mac OS 8. Wherever graphics code was redundant, it has been combined in Mac OS 8, decreasing code size and the attendant memory requirements for running more than one graphics system simultaneously.

The built-in QuickDraw 3D capabilities provide an intuitive user interface for manipulating three-dimensional objects and allow applications to share three-dimensional data easily, even across the Internet. Mac OS 8 printing is based on the QuickDraw GX model familiar to many System 7.5 users—to print, users drag document icons to desktop printer icons. ColorSync technology is also built in, so colors rendered by a printer match those displayed on the screen. The line-layout capabilities of QuickDraw GX are also integrated into the system. Mac OS 8 provides such formatting features as kerning, ligatures, and the multiple reading directions—such as vertical and right-to-left—needed for such writing systems as Chinese, Japanese, Arabic, and Hebrew.

The multimedia capabilities of QuickTime, QuickTime VR, and QuickTime Conferencing are also incorporated into Mac OS 8. QuickTime supports the editing, storing, and playing of synchronized data such as video and audio. With QuickTime VR, users can take virtual reality tours of distant museums and landmarks, view ideas for a kitchen remodel, and visit imaginary places. Through QuickTime Conferencing, individuals can exchange voice and data simultaneously over telephone lines, local area networks, and the Internet.

OpenDoc

Mac OS 8 incorporates OpenDoc, an industry standard technology based on the concept of component software—that is, self-contained, reusable software modules. OpenDoc environments are also under development for the Windows, OS/2, and UNIX operating systems. With OpenDoc, users can add or remove a feature by dragging it into a document or workspace. A user can combine favorite features, such as tools for handling text, graphics, photography, spreadsheets, and video—even live data links and Internet connections—for use in a single, customized work environment.

Through OpenDoc, Mac OS 8 allows individuals and organizations to choose the specific features they need without learning and supporting multiple applications. At the same time, OpenDoc allows organizations to standardize their development efforts on a cross-platform architecture for both commercial and custom software.

Communications and Networking

With the AppleTalk software and hardware architecture, Apple Computer introduced plug-and-play networking capabilities for personal computers. To broaden the networking capabilities of personal computers, Mac OS 8 incorporates support for a wide array of communications standards through its Open Transport and Cyberdog architectures.

Open Transport, available for all Mac OS-compatible computers since the release of System 7.5.3, supplies an architecture for implementing industry-standard communications and networking protocols. To incorporate networking and communications services in an application, regardless of the type of network to which the user's computer is connected, a developer uses the programming interface defined by Open Transport. Open Transport then takes care of the communication details appropriate for the user's network. Open Transport includes implementations of the AppleTalk and TCP/IP protocols and support for common data links such as LocalTalk, Ethernet, and Token Ring.

The Cyberdog architecture gives users a consistent, intuitive way to search and browse the Internet and gain access to Internet mail and newsgroups. This architecture supplies a collection of OpenDoc classes and extensions with which developers create OpenDoc parts that access and display Internet content. Cyberdog, in other words, is a toolbox for supplying Internet capabilities, such as web browsers and e-mail facilities. Mac OS 8 includes several Cyberdog parts with which users and developers can instantly create OpenDoc documents that have embedded Internet functionality. To support Internet capabilities within their applications, developers need simply support OpenDoc, but the Cyberdog architecture also makes it easy for developers to create their own parts and to replace the Mac OS 8–supplied parts.

Mac OS 8 supplies other user-level networking features. Personal file sharing—which allows any Mac OS–compatible computer on a network to be a file server—and network printer sharing were introduced in previous versions of the Mac OS. These are incorporated into Mac OS 8, which introduces other user features designed with networking in mind. The workspaces feature, for example, allows several people to maintain separate networking preferences, such as e-mail accounts and World Wide Web bookmarks, on the same computer. Or, one person can use several workspaces to maintain several personal accounts. With the Assistance Services, Mac OS 8 supplies a network configuration expert to help users connect to networks. And, as previously described, with Find windows users can locate information on networks and save the locations of found information. Other Mac OS 8 technologies—such as multitasking, memory protection, and system-level Unicode support—further increase the performance of network and communications products available from developers. For instance, personal Internet server programs on Mac OS 8 computers benefit from the reliability of memory protection, the performance of preemptive multitasking, and the flexibility of international language support.

SUMMARY

By supporting computers from multiple manufacturers and allowing users to personalize their work environments according to their needs and tastes, Mac OS 8 delivers an extremely flexible computing platform. New ease-of-use features and state-of-the-art operating system services help users be more productive in their professional, educational, and recreational pursuits.

The capabilities of Mac OS 8 briefly presented in this chapter—and many additional capabilities—are explained in more detail in the rest of this book. Read on for more information about how developers can take advantage of the Mac OS 8 architecture in their products.

2 Orientation to the Mac OS 8 Platform

This chapter offers a high-level tour of the Mac OS 8 platform from the developer's perspective. Capable of supporting a wide variety of high-performance hardware, the Mac OS 8 operating system provides a platform for new software products that increase user productivity. These new products can take advantage of such features as a flexible, consistent, and easy-to-use human interface, preemptive multitasking, concurrent I/O processing, memory protection, software extensibility, automated user assistance, large amounts of addressable memory, and enhanced versions of such established Mac OS technologies as graphics, multimedia, networking, and collaboration.

Mac OS 8 not only supports these innovations but also provides compatibility for most applications developed for System 7. This backward compatibility allows users to protect much of their investment in System 7 software even while moving forward to take advantage of the new capabilities available in Mac OS 8.

Read this chapter to become acquainted with the operating-system services of the Mac OS 8 platform.

KEY TERMS AND CONCEPTS

- The **operating system** is the software that controls and coordinates computer hardware and supports the execution of application-level programs installed or controlled by users.
- A **reentrant service** is a Mac OS 8 operating system facility that can be used concurrently by several pieces of code. These services allow many concurrent operations to be preemptively scheduled for execution. The microkernel and the I/O system are examples of reentrant services.
- The **microkernel** is a program that manages a small but critical subset of the operating services necessary to control the computer. In essence, the Mac OS 8 microkernel manages the computer resources (such as memory, synchronization, timing, and messaging) necessary for code to execute on the CPU. Other operating system services, such as the I/O system and the Human Interface Toolbox, are implemented separately from the microkernel.
- A **cooperative service** is used by programs that cooperate with each other to synchronize their access to the service. A program generally uses cooperative services to present a human interface. Cooperative services also support compatibility with System 7 applications.
- The **human interface** provides elements allowing the user to interact with programs running on a computer. Because most human interface elements (such as windows, menus, and icons) are visual in the Mac OS, the term *human interface* is generally synonymous with **graphical user interface**. However, user voice input, sounds that alert the user, and other nonvisual elements are part of the human interface as well.
- A **cooperative program** typically presents a human interface and cooperates with other programs to share access to the cooperative services. Cooperative programs are usually implemented as interactive applications or compound documents containing OpenDoc parts.
- A **server program** in Mac OS 8 is a program that has no direct interaction with users but instead provides an offscreen service, usually to one or more other programs—for instance, by performing calculation-intensive or I/O-intensive operations on behalf of cooperative programs.
- **Preemptive scheduling** is a policy for allocating access to the CPU and other computer resources. This policy allows the operating system to preempt the execution of one operation and start—or resume—the execution of another. All server programs are preemptively scheduled for execution, and portions of cooperative programs unrelated to the human interface can be preemptively scheduled for execution. All coop-

erative programs are, by comparison, cooperatively scheduled for execution.

- **Cooperative scheduling** is a policy for serializing cooperative program access to the cooperative services. Under this policy, every call to a cooperative service executes to completion without the possibility of interruption by another call to that same service. Cooperative programs participate in this policy by yielding execution eligibility to one another, ensuring that no more than one program at a time is able to call the cooperative services. Cooperative scheduling rotates eligibility among cooperative programs so that each can, in turn, be preemptively scheduled along with server programs.
- An **application** is a program designed to help users accomplish goals; for example, a web browser helps users navigate the World Wide Web and a page-layout program helps users present information in print form. In Mac OS 8, an application is usually implemented as a cooperative program that may be supported by server programs.
- **OpenDoc** is a multiplatform technology for constructing and sharing compound documents. Compound documents consist of multiple, user-selected software components (called **parts**), which are used for creating, containing, and displaying information.

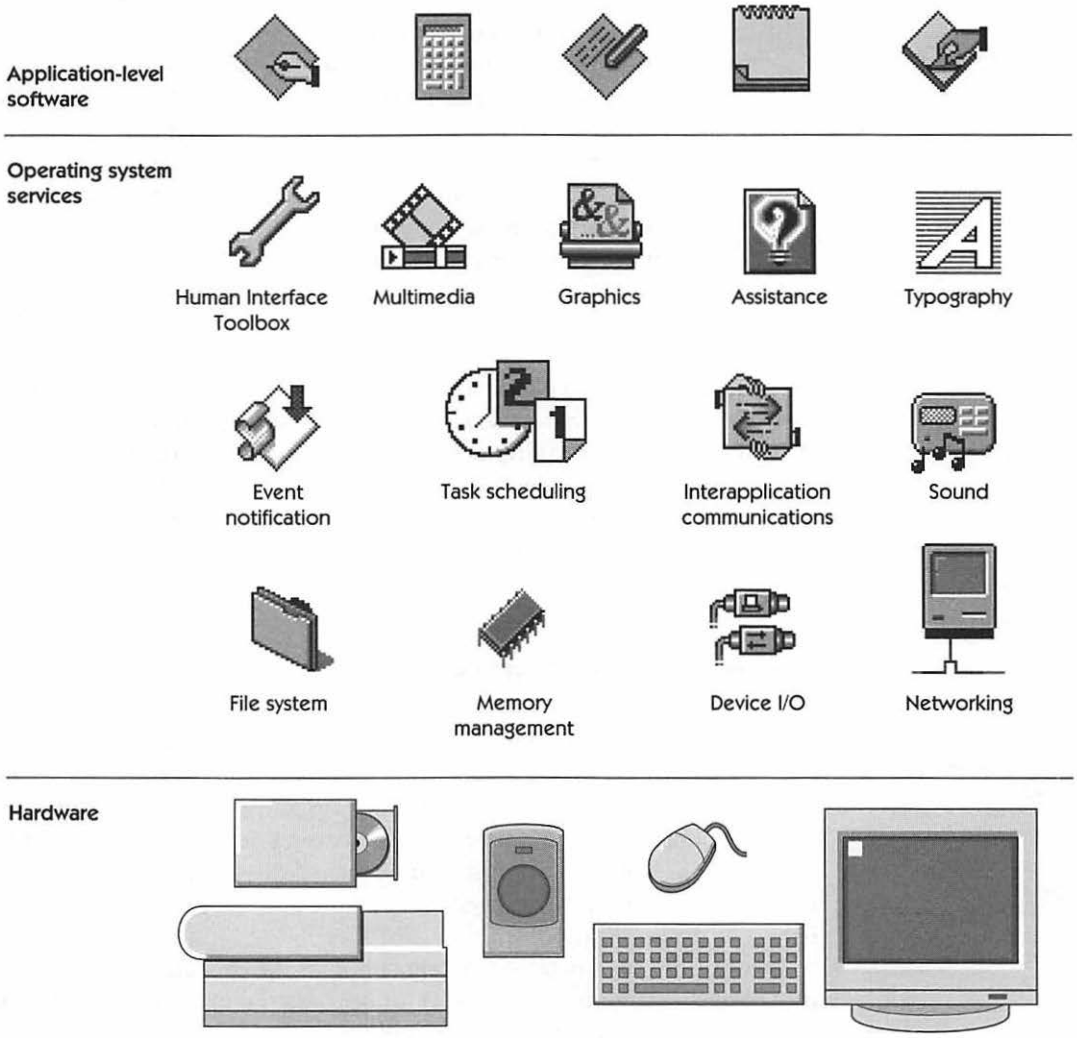
MAJOR POINTS OF INTEREST

A **program** is a series of statements instructing a computer to perform certain operations.

Users interact with the Mac OS 8 platform through its hardware and its application programs. Using a mouse and keyboard, for example, a user manipulates the information calculated and organized by an application and displayed onscreen. The application, however, doesn't directly manipulate the video screen, nor do the mouse and keyboard directly manipulate the application. Instead, the operating system controls and coordinates all hardware on behalf of the application.

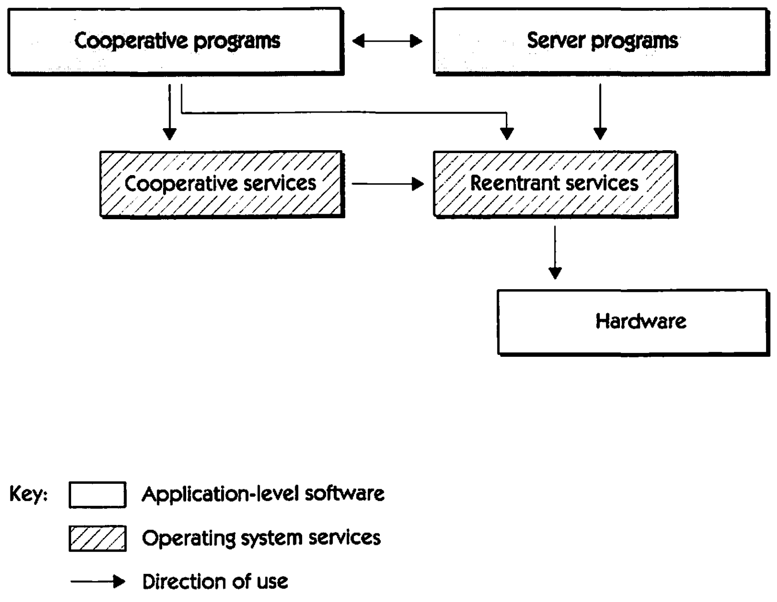
In addition, the operating system supports the execution of applications by providing them with a wide array of services. Figure 2.1 illustrates some of these services. Developers, for example, use the Human Interface Toolbox to implement the interactive features of their applications; they invoke assistance services to provide users with automated help; they call event notification services to determine when and how to respond to user actions and various system activities; they use task scheduling services to take advantage of the system's multitasking and multithreading capabilities; they call file system services to store and retrieve user data; and they employ networking services to help users collaborate and communicate via computers.

FIGURE 2.1 Major components of the Mac OS 8 platform



The Mac OS 8 platform is designed to be modular. Modularity allows the platform to be adapted in the present and enhanced in the future by Apple Computer and other developers. For example, the operating system is abstracted from the hardware; therefore, computer manufacturers can more easily port Mac OS 8 to their various hardware architectures. Application-level software is abstracted from the operating system; therefore Apple Com-

FIGURE 2.2 Reentrant and cooperative services on the modular Mac OS 8 platform



puter can enhance operating system services without adversely affecting the programs that rely on them.

Figure 2.2 diagrams the broadest elements of the Mac OS 8 platform's modular design. As this figure shows, the operating system comprises two major types of services: reentrant services and cooperative services. Generally speaking, developers use the reentrant services to take advantage of the increased efficiency afforded by concurrent processing and preemptive scheduling. Developers use cooperative services to present a human interface for their programs.

Concurrent processing allows separate programs to share operating system services in parallel.

Application-level software consists of cooperative programs and server programs. Cooperative programs, such as interactive applications and compound documents containing OpenDoc parts, present a human interface. Cooperative programs can use both cooperative and reentrant services. Server programs can use the reentrant services but not the cooperative services. Server programs don't present a human interface; instead, they perform work offscreen, often on behalf of cooperative programs. To interact indirectly with the user, however, a server program can use a reentrant service to send user notifications (such as sounds, icons that blink at the top of the screen, and short onscreen messages), and a server program can employ a cooperative program to perform user interactions on its behalf.

The rest of this chapter examines more closely the elements diagrammed in Figure 2.2.

THE HARDWARE

The Mac OS 8 platform supports

- computers, produced by a variety of computer manufacturers, that are compliant with the Common Hardware Reference Platform (CHRP)
- models of Apple Power Macintosh, Macintosh PowerBook, and Apple Performa computers with PowerPC-based logic boards
- PowerPC-based computers from Apple-licensed manufacturers

Physical memory, also known as RAM, holds program code and data temporarily needed by the CPU.

All of these computers are shipped with at least 8MB of physical memory, the minimum targeted by Apple Computer for Mac OS 8.

A Mac OS 8–compatible computer typically employs a keyboard and mouse for user input and a display screen for program output and usually includes stereo sound through built-in or separate speakers, a built-in networking device, and additional devices such as a modem, CD-ROM drive, and microphone.

At its most fundamental level, the hardware layer of the Mac OS 8 platform rests on the PowerPC family of CPUs. Above this level, the hardware layer is highly flexible. The modular design of Mac OS 8 allows developers to extend or differentiate the platform for different hardware products. Computer manufacturers, for example, can incorporate such diverse bus architectures as SCSI, PCI, and FireWire into the I/O system, and peripheral-device developers can easily integrate such products as video capture devices, scanners, graphics tablets, laboratory equipment, and remote infrared devices into the system.

A **programming interface** consists of functions and data structures defined by one piece of software, such as an operating system service, for use by client software, such as applications and device drivers. The Mac OS 8 programming interface provides access to such services as window management and file management.

THE OPERATING SYSTEM

Application-level software makes use of operating system services, both reentrant and cooperative, through their programming interfaces. As you can see in Figure 2.2 on page 29, application-level software is insulated from the hardware by these services. Various reentrant services control the operation of hardware devices (including display screens, network devices, hard disks, and modems) to support the execution of application-level software. Not all reentrant services control hardware, but virtually all hardware is controlled by reentrant services. (The only exception is that Mac OS 8 allows applications with special needs to draw directly to the frame buffers for video devices.)

In addition to the operating system services described next, Mac OS 8 usually includes a number of other applications that aren't strictly part of the operating system but nevertheless enhance user productivity—for example, a Mac OS 8 system may include such programs as the Finder, the SimpleText text processor, and the Personal FileShare file server program. Apple Computer and other computer manufacturers often pre-install additional game and productivity programs on their computer systems as well.

Concurrent Processing and the Reentrant Services

Many Mac OS 8 operating system facilities are implemented as reentrant services to support the concurrent processing of multiple programs. **Reentrancy** is the ability of code to process multiple interleaved requests for service nearly simultaneously. For example, a reentrant function can begin responding to one call, become interrupted by other calls, and complete them all with the same results as if the function had received and executed each call serially. (A function that isn't reentrant must complete one call before receiving another.)

The code for each reentrant service is written so that it synchronizes all access to its data, thereby allowing programs to call the service *at any time* without the risk of corrupting the data used by that service. Multiple programs, then, can call a reentrant service concurrently, allowing the operating system to schedule these programs preemptively—that is, the operating system can suspend the execution of one program and allow another to execute, even if both programs make requests to the same service.

Preemptive scheduling makes efficient use of the CPU by keeping it as busy as possible. For example, the CPU executes instructions faster than I/O devices transfer data. So when a program uses a reentrant service to read data from a disk drive, the operating system will suspend the program when its read operation is waiting for data to come off the disk. The operating system then schedules other operations for execution, keeping the CPU busy even while the program is waiting for the arrival of data from the disk. At the same time, other programs can concurrently perform I/O operations of their own, which are interleaved in this efficient, preemptively scheduled manner.

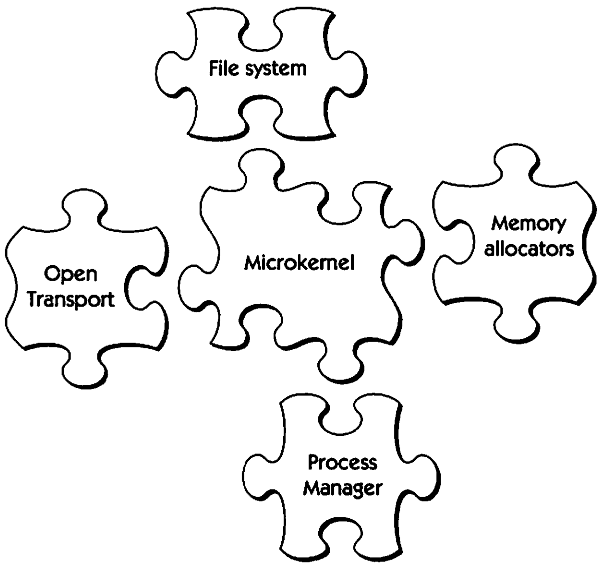
Preemptive scheduling is described in greater detail in **Chapter 4**.

Reentrant services include

- the microkernel
- the I/O system, including the Mac OS 8 file system, the Open Transport networking services, and real-time sound playback
- the memory allocation services
- the Apple events interprocess communication service
- the QuickDraw GX graphics system

These services are described at greater length in later chapters.

FIGURE 2.3 The microkernel and other modularized reentrant services



The microkernel lies at the core of the operating system. The microkernel, in essence, manages the computer resources necessary for code to execute on the CPU.

Kernels in mainframe, minicomputer, and workstation platforms traditionally encompass all or most operating system services. The current trend in operating system design is to implement a subset of essential operating system services in a smaller kernel, a so-called microkernel. By moving various operating services out of the kernel, a microkernel-based operating system becomes more modular and flexible than one that is kernel-based.

In a UNIX-based operating system, for example, the kernel is a program that supervises task and file management, device input and output, and memory allocation, whereas the microkernel design of Mac OS 8 modularizes these services, as illustrated in Figure 2.3. This modularity makes it easier for Apple Computer and other developers to update and extend the operating system. For example, Apple Computer can in future releases of Mac OS 8 easily add or replace memory allocators as the system's dynamic storage needs evolve. Because the I/O system is further modularized, computer manufacturers can differentiate their products in a wide variety of ways, such as by supplying test equipment for laboratories or devices optimized for video data. Such products are easily integrated with the I/O system and, because of the modular nature of the entire operating system, work without any modification to the microkernel itself.

MAC OS HERITAGE

Application Control of the Computer in Previous Mac Operating Systems

In System 7 (and earlier versions of the Macintosh Operating System), all operating system services are provided by libraries of routines (called **managers** in Mac parlance) that applications call at points chosen by their developers. Before Mac OS 8, the Macintosh Operating System used neither a kernel nor a microkernel to control access to computer resources. Instead, applications largely had complete control of the computer.

Serialized User Interactions and the Cooperative Services

Mac OS 8 developers use the cooperative services to incorporate a human interface into their programs. From among these services, for example, developers use the **Human Interface Toolbox** to implement windows, menus, and other portions of the standard graphical user interface, and developers use the **Assistance Services** to provide interactive and automated help for users. In addition to supporting human interface features, the cooperative services maintain compatibility with applications written for System 7. Cooperative services include

- the Human Interface Toolbox
- the Assistance Services
- the QuickDraw and QuickDraw 3D graphics systems
- the QuickDraw GX printing service
- multilingual text processing services
- the QuickTime multimedia technologies, including virtual reality and conferencing capabilities
- additional compatibility services to support System 7 applications, such as the System 7 File Manager and the System 7 Memory Manager

These services are described at greater length in later chapters.

Whereas the reentrant services can be called by several pieces of software concurrently, the cooperative services can't. To synchronize access to the cooperative services and provide compatibility for System 7 applications in a preemptively scheduled environment, Mac OS 8 employs another scheduling policy, called cooperative scheduling. This policy is supervised by an operating system service called the **Process Manager.** Whereas a reentrant service synchronizes multiple requests, a cooperative service relies on its clients to serialize their requests so that the service can finish one request before receiving another. When programs cooperate by yielding execution eligibility to one another, the Process Manager serializes their calls to the cooperative services.

By serializing calls to the cooperative services, the Process Manager allows each call to a cooperative service to execute to completion without being interrupted by another call to the same service. A Mac OS 8 application adheres to this cooperative policy by yielding execution eligibility whenever there are no events (such as keystrokes, menu selections, or operating system communications) for it to respond to. By following the System 7 event-handling model, System 7 applications and OpenDoc parts are fully compatible with this policy.

Application event handling is explained in **Chapter 14**.

When an event arrives for a cooperative program, it becomes eligible for execution. That program is then preemptively scheduled with other eligible tasks. Other eligible tasks may include those for server programs and the operating system itself. In addition to one cooperatively scheduled task, a cooperative program can include additional, preemptively scheduled tasks to enhance user productivity, but these tasks can't use any cooperative services.

The use of multiple preemptively scheduled tasks within a program is described in **Chapter 5**, and the relationship between cooperative scheduling and preemptive scheduling is further delineated in **Chapter 4**.

Serializing Operations Involving User Interactions

In program operations that involve interaction with the user through the screen, a cooperative approach involving serial access makes sense. From the user's perspective, one screen operation shouldn't be preempted by another screen operation while the first operation is still underway. If the user has two applications running, for example, it wouldn't be sensible for the background application to create a new window on top of the window in which the user is currently typing text. Similarly, when the user selects a menu command from one application, it would be confusing if another application were to interrupt the command and perform some onscreen action of its own.

System 7 developers are familiar with a programming model where applications share processing time but only one application is the focus of user interaction. Mac OS 8 developers also follow this general model for programs involving user interaction. For example, the QuickDraw GX graphics system is implemented as a reentrant service. This implementation allows developers to perform image processing operations offscreen where, outside the cooperative scheduling environment, they take maximum advantage of preemptive scheduling. However, when developers use QuickDraw GX to draw to the screen, they do so exclusively from their cooperative programs. Confining all screen drawing operations to the cooperative scheduling environment ensures that graphical operations involving user interaction are properly serialized.

System 7 Application Compatibility

Apple's goal is to make more of its operating services reentrant in future versions of Mac OS 8 so that preemptively scheduled operations, such as those involving server programs, might be able to use portions of the Human Interface Toolbox and other such services that require cooperative scheduling today. In the meantime, Apple decided not to implement the cooperative services in reentrant form because, as you've read, human interface operations need to be serialized in some manner. Cooperative scheduling simplifies the programming required by developers to serialize these operations, and System 7 developers are already familiar with this policy. Moreover, Apple wanted to maintain backward compatibility with System 7 applications, which are designed to rely on cooperative scheduling.

With every Macintosh Operating System release, Apple has protected user investments in software and hardware by ensuring a high degree of backward compatibility. Many cooperative services are available in Mac OS 8 only to provide backward compatibility for System 7 applications. For example, Mac OS 8 supports the System 7 File Manager programming interface as a cooperative service even though Mac OS 8 supplies a new, fully reentrant file system. Developers can use either service for performing file management in their cooperative programs, but the reentrant service provides much better system performance.

System 7 applications have additional compatibility dependencies that cannot be solved simply by replacing the cooperative services with reentrant versions. In particular, System 7 applications rely on a single address space shared by all other user-interactive applications running in the system. (For now, suffice it to say that an address space is the range of memory visible to a program that is executing.) As you will see in Chapter 3, Mac OS 8 supports multiple address spaces. When System 7 application compatibility becomes less critical to users and developers, Apple can move to an operating system where every application resides in its own protected address spaces. Programs written for Mac OS 8 will automatically take advantage of these features when they become available.

Using Preemptively Scheduled Operations from a Cooperative Program

In the cooperative scheduling environment of Mac OS 8, a program calling the cooperative services can't be preempted by any other program calling the cooperative services. However, a program calling the cooperative services *can* be preempted by a program calling only the reentrant services, which are implemented in areas—such as the file system and the rest of the I/O system—where preemptive scheduling offers significant gains in system efficiency. A developer with an I/O-intensive or computation-intensive application can improve system efficiency and user productivity by dividing the application into one or more preemptively scheduled portions and a cooperatively scheduled portion that handles user interaction. As explained in Chapter 5, one portion of a scientific simulation application might perform preemptively scheduled statistical calculations in the background while another portion, cooperatively scheduled, allows the user to work with the program's human interface.

APPLICATION-LEVEL SOFTWARE

As you've seen in the previous discussions, the operating system exists largely to support application-level software—the programs that allow users to accomplish goals. These goals can be as diverse as managing a business, producing a music video, and getting an up-to-the-minute ski report.

Mac OS 8 supports two broad classes of application-level software:

- cooperative programs, with which users interact
- server programs, which are invisible to users but nonetheless work very efficiently behind the scenes, often on behalf of cooperative programs

Cooperative Programs

Cooperative programs are those with which the user interacts—for instance, by using the mouse and keyboard to manipulate windows, menus, images, text, and other onscreen items. Examples of cooperative programs include

- document processors
- digital video editors
- World Wide Web browsers
- OpenDoc documents
- games

To present a human interface, a cooperative program uses the cooperative services. A cooperative program can also use the reentrant services, as illustrated in Figure 2.2 on page 29. Notice in this figure how cooperative programs are insulated from the hardware by the operating system services. This insulation from the hardware allows software developers to create programs without concerning themselves with the underlying hardware.

A cooperative program must make all requests to the cooperative services from its **main task**—that is, its initial path of execution. Without any modification, System 7 applications and OpenDoc parts already adhere to this design rule. When the developer of a new interactive program adheres to this rule while structuring the program to support cooperative scheduling, the Process Manager automatically synchronizes the program's access to the cooperative services.

Chapter 14 explains how a developer structures a Mac OS 8 program to handle events in a way that supports cooperative scheduling.

The OpenDoc environment also ensures that the Process Manager synchronizes access to the cooperative services for all OpenDoc parts contained in an OpenDoc document. Because the part editors within an OpenDoc document handle user interaction, they use cooperative services. The main task for an OpenDoc document incorporates the document's constituent part editors, and this main task is cooperatively scheduled like the main task of any other cooperative program. Throughout the rest of this book, whenever the term **cooper-**

ative program is used, you should read it to mean an application program with a user interface or a document containing various OpenDoc parts.

MAC OS HERITAGE

OpenDoc

OpenDoc technology became available on the Mac OS in November, 1995. OpenDoc development is also underway for the Windows, O/S 2, and AIX platforms, and it has been tightly integrated into Mac OS 8. OpenDoc centers around user-extensible documents. Users buy or create compound documents, in which collections of software components, called **part editors**, replace monolithic applications. Each part editor is responsible for manipulating specific types of content within a portion, or part, of a compound document.

The user doesn't launch part editors directly. Instead, the user works with the various parts of an OpenDoc document, which acts like a shell to hold the various parts. As the user works with a part, the code for that part editor runs and manipulates the data within the part.

Users can add or remove a part just by dragging it in or out of a compound document. For example, a student preparing an online lab report might wish to include video images of a physics experiment. The student could use the mouse to drag a video part into the document to incorporate an illustrative video. The student could likewise add a part that prepares three-dimensional graphs of laboratory data. As more ideas arise, the student could further extend the capabilities of the document by embedding additional parts.

The modular nature of the Mac OS 8 platform gives developers a great deal of flexibility in creating products that integrate easily with the operating system, and this modularity gives users a great deal of flexibility in choosing and using different configurations of this platform. In a similar way, the modular nature of OpenDoc gives developers a great deal of flexibility in creating products that integrate seamlessly with users' personally configured work environments.

Server Programs

In a **client/server relationship,** computing operations are split between two entities: clients, which request services, and servers, which provide services.

In Mac OS 8, server programs provide background processing services, generally to client programs. Unlike cooperative programs, server programs in Mac OS 8 can have no direct interaction with users. A server program can have one program as a client or many, on the same computer or on remote computers connected to a network.

Examples of server programs include

- World Wide Web servers
- file-compression utilities
- e-mail servers
- calculation engines for data-intensive statistical simulations
- database servers

Unlike cooperative programs, which typically launch and quit under user control, server programs generally launch automatically when they're needed and run until they're no longer needed. When a developer wants to create a program that can start and quit automatically at any time and requires no user interaction, a server program is a logical implementation choice. For example, a file-compression utility that saves disk space by automatically compressing files that are a certain number of days or weeks old could be implemented as a server program. (As you will see in Chapter 13, a developer could use file modification dates as conditions that trigger the launching of this file-compression utility.)

Unlike cooperative programs, which share a single address space in a manner that supports System 7 application compatibility, every server program operates on data in its own protected address space. (You'll learn more about Mac OS 8 address spaces in the next Chapter 3.) A program that greatly benefits by having its data protected within its own address space is best implemented as a server program. For example, a database server that supplies information important to the minute-to-minute operations of a company would be suitably implemented as a server program.

A developer can design a single server program to satisfy the needs of multiple clients. For instance, a server that performs data-intensive statistical simulations could be used by a suite of cooperative programs—such as a scientific simulation program, an engineering design program, and a three-dimensional rendering program. And as this example suggests, server programs can use cooperative programs to present a user interface. As you'll see in Chapter 5, Mac OS 8 provides interprocess communication mechanisms that allow a server program and a cooperative program to exchange information and to direct each other in the manipulation of that information.

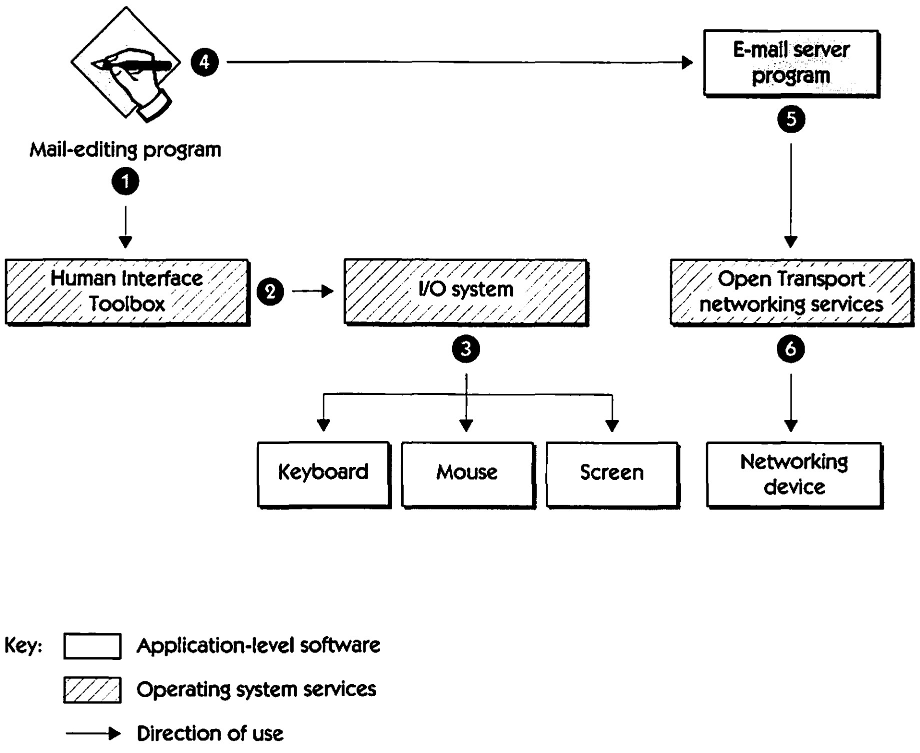
Figure 2.4 illustrates how a cooperative program and a server program might interact. In this figure, a mail-editing program uses the Human Interface Toolbox, a cooperative service, to present a human interface so that the user might create and read e-mail messages. The Human Interface Toolbox uses reentrant services of the I/O system to receive user input through the keyboard and mouse, and to determine various screen characteristics when displaying information onscreen. By manipulating the keyboard and mouse to interact with the mail-editing program, the user composes, addresses, and sends an e-mail message.

When the user directs the mail-editing program to send a message, the program uses a separate e-mail server program to deliver the message over a network. The e-mail server program uses the reentrant Open Transport networking services to handle the network communications; Open Transport, in turn, manages the hardware networking device.

Open Transport and related networking facilities are described in **Chapter 16**.

The server program can also make use of the cooperative program to display incoming messages. In this scenario, the e-mail server program employs Open Transport to listen to network traffic for e-mail. When a message

FIGURE 2.4 Interactions involving a server program and a cooperative program



arrives, the e-mail server program forwards the message to the mail-editing program. The mail-editing program then interacts with the user by employing the Human Interface Toolbox and other cooperative services to draw the text of the message inside a window, which is displayed on the user's screen.

SUMMARY

The modular nature of the Mac OS 8 platform offers these advantages to developers:

- Because the operating system is abstracted from the hardware, Apple Computer and other developers can easily adapt Mac OS 8 to work with many different types of hardware, allowing hardware manufacturers to develop well-differentiated products.

- The microkernel and other operating system services are modularized, allowing Apple Computer and other developers to incorporate adaptations and enhancements to the operation system with less effort.
- Because application-level software is abstracted from the operating system, Apple Computer and other developers can easily adapt the operating system without disrupting programs that use its programming interface.

The operating system offers reentrant services to programs in areas where the system can maximize efficiency through the concurrent processing of multiple, preemptively scheduled tasks. The operating system offers cooperative services to programs that present a human interface. Access to these services is cooperatively scheduled so that all calls to these services are serialized. In addition to human interface services, other cooperative services are supplied to maintain compatibility with applications written for System 7.

Software products written to present a human interface are executed as cooperative programs. Software products written exclusively to provide off-screen services are executed as server programs. A server program operates within its own address space, where its data is protected from inadvertent access by other programs. A server program presents no user interface but can use cooperative programs and operating system services to interact with the user.

A developer can create an application that combines a cooperative program to perform user interaction onscreen with a server program that performs operations in the background. This combination allows the user to work productively with one part of the application even while another portion performs preemptively scheduled I/O-intensive and calculation-intensive operations. As you'll see in Chapter 5, a developer can also incorporate these same preemptively scheduled operations in a cooperative program by creating additional tasks for that program.

3

Address Spaces and Memory Protection

When a program is launched—for instance, when a user double-clicks its icon—the operating system prepares the program code for execution, creates memory areas for the code and its temporary data, and assigns locations for the code and data within these memory areas. In this way, the program becomes instantiated as a process on the computer. The memory areas created for a process lie within a 4-gigabyte (GB) range of logical addresses. This range of addressable memory constitutes the address space for that process.

Mac OS 8 maintains multiple simultaneous address spaces. A program can't reference any memory locations outside of its address space. Therefore, if code in a given address space malfunctions, it can't corrupt the data in a different address space. Mac OS 8 provides other forms of memory protection, too. Mac OS 8 protects all code, for example, by mapping it into read-only memory areas where it can't be corrupted by any errant code elsewhere in the system. Crucial system data is protected because it's stored in memory areas where operating system services—such as the microkernel, device drivers, and the file system—have read/write permission to the data, but application-level software has read-only permission. This greatly decreases the ability of applications to cause a system-wide crash. Yet another kind of memory protection, called **guard pages,** enhances system stability by limiting the amount of damage that software can do if it attempts to read or write outside the memory area it's entitled to access.

KEY TERMS AND CONCEPTS

- A process is an instance of a program running at execution time. A process is characterized by a set of one or more tasks and the operating system resources necessary to support those tasks.
- A **task** is the basic unit of program execution in Mac OS 8. Every process has at least one task. As you'll read in the next chapter, each task is assigned a priority and, when eligible for execution, is preemptively scheduled by the microkernel.
- A **memory area** is a range of logical addresses.
- **Virtual memory** is addressable memory beyond the limits of available physical memory. Mac OS 8 extends physical memory by storing on a secondary storage device, such as a hard disk, code and data not immediately required by the CPU.
- A **logical address** is a memory address used by code when it's running. By comparison, a **physical address** is a memory address represented by bits on a physical address bus. Physical addresses are assigned to memory locations in RAM chips and to various hardware devices. When executing code, the CPU translates the logical addresses of an address space into physical addresses.
- An access **permission** stipulates whether other programs can read from or write to a memory area.
- A **guard page** is a 4-kilobyte (K) range of logical addresses that excludes all program access. Guard pages may appear at the beginnings and ends of memory areas to help prevent code from inadvertently accessing the wrong memory areas. If a programming error causes code to reference a guard page, the CPU generates an exception before the erring code can adversely affect a contiguous memory area.

MAJOR POINTS OF INTEREST

All code and data for a process exist within an address space. Because Mac OS 8 uses a 32-bit address space—which is the maximum size supported by the PowerPC CPU—an address space can contain up to 2^{32} addresses. In every address space, in other words, addressable locations number up to 4GB.

A 4-GB address space encompasses far more memory addresses than are available in physical memory on most computers. So Mac OS 8 uses a virtual memory system to extend the range of addressable memory beyond what is available in physical memory. The virtual memory system stores unused portions of code and data on a secondary storage device, such as hard disk. The virtual memory system then transfers into physical memory only those portions immediately needed by the CPU. (As you'll see in Chapter 6, the virtual memory system also makes efficient use of secondary storage by using only enough disk space to support currently open programs.)

When launching a program, the operating system creates memory areas that constitute only a small portion of an address space. The operating system creates a memory area for the program code, and it creates an initial memory area for program to store the data—such as its global variables and dynamic data structures—that it needs while it's running. Other portions of an address space are unavailable to the program because they're used to store code (including code for the microkernel and code for the libraries used by the program), or they're reserved for other uses by the operating system. From the 4GB of logical addresses in a single address space, at least 1GB is available to programs for data storage.

As you'll see in Chapter 7, the operating system dynamically creates and releases memory areas as needed so that programs can store temporary data. The Dynamic Storage-Allocation Services provided by Mac OS 8 also allow developers to create their own memory areas suitable for special program needs.

For overall system stability, Mac OS 8 employs multiple address spaces. The data referenced by a program in one address space is inaccessible to programs in other address spaces. Therefore, programming errors affecting one address space are isolated from all other address spaces. For example, suppose that a game program has a programming error that corrupts portions of its address space, causing the game to crash. Operating on data in its own address space, a World Wide Web server program continues serving web pages, immune to the game's error.

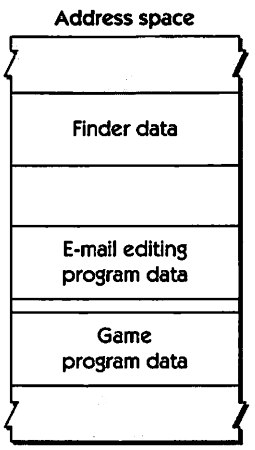
Within an address space, areas of memory may be further protected by access permissions. For example, all executable code in Mac OS 8 is stored in read-only memory areas where code can't possibly be corrupted. And data used by critical portions of the operation system, such as the microkernel, is kept in areas protected by access permissions that prevent applications from corrupting it.

For compatibility with System 7 applications, which rely on a single address space, all cooperative programs share a single address space. Every server program, by comparison, is given its own address space.

THE COOPERATIVE PROGRAM ADDRESS SPACE

Whereas Mac OS 8 supports multiple address spaces, System 7 supports only one address space. To provide compatibility for System 7 applications, many of which are designed to read or manipulate each other's data structures, Mac OS 8 assigns all cooperative programs to a shared address space. Figure 3.1 illustrates the cooperative-program address space for a system on which

FIGURE 3.1 Cooperative programs sharing an address space



the user has launched an e-mail editing program and a game program from the Finder program. All three cooperative programs store their temporary data in this address space. (These applications, by the way, are cooperative programs because they present a human interface.)

The figures in this book don't literally represent the layout of logical memory. For example, data for the Finder appears near the top of the address space in Figure 3.1; however, Finder data isn't necessarily mapped into memory areas at the top of the cooperative address space.

Whereas the amount of memory that's available to applications in System 7 is usually far less than 4GB, an entire 4-GB address space is available to them in Mac OS 8. This large amount of addressable memory, backed by the Mac OS 8 virtual memory system, allows the user to keep many more applications open simultaneously than is possible in System 7.

Like Mac OS 8, System 7 uses a 32-bit address space, where any address between 0x0000 0000 and 0xFFFF FFFF is a valid logical address. In System 7, however, the range of logical addresses actually available from this address space is determined at system startup by the amount of virtual memory previously selected by the user. Mac OS 8, by comparison, dynamically allocates storage locations from this address range to satisfy program needs as they arise.

For example, if a user in System 7 sets total memory to 12MB and launches an e-mail application and a game, they'd share 12MB of addressable memory even if they required only 5MB between them. If the user then tried to launch a photo-editing application requiring 8MB of addressable memory, the program would fail to open because of insufficient memory. To launch the photo-editing program, the user would need to quit the e-mail application or the game.

When these same programs are launched in Mac OS 8, the operating system supplies their memory needs dynamically. For example, the operating system allocates from the 4-GB address space only the 5MB necessary to run the e-mail program and the game. When the user launches the photo-editing application, the operating system allocates another 8MB from this address space. As the user launches more applications, Mac OS 8 continues allocating more addressable memory from the address space. (As you'll see in Chapter 6, the number and size of applications that the user may launch are constrained only by the disk space available to the virtual memory system for storing temporary data. To extend virtual memory without consuming any additional disk space, the operating system memory-maps the disk files of all code used at execution time.)

The enormous range of addressable memory that Mac OS 8 supplies to cooperative programs nearly eliminates the memory fragmentation problems experienced by users of operating systems supplying smaller amounts of addressable memory. For example, a System 7 user might launch enough applications to fill all 12MB of available memory and then quit two applications to release 8MB of memory. If the two applications weren't contiguous in memory, the total available memory might be fragmented into two 4-MB areas, preventing the user from launching a 5-MB application. On a Mac OS 8 system, memory for this application would be allocated from some unused portion of the 4-GB address space.

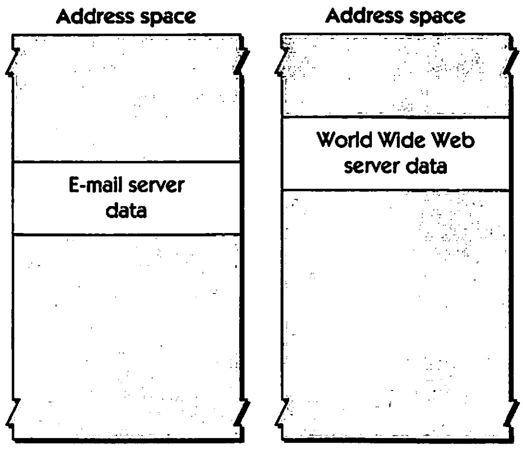
PROTECTED ADDRESS SPACES FOR SERVER PROGRAMS

When a server program is launched (usually this happens automatically when the user starts the computer), the operating system instantiates the process for that server program in its own address space. Because every server program exists in its own address space, where other programs can't address its data, server programs are protected from possible programming errors in cooperative programs and other server programs.

Figure 3.2 illustrates separate address spaces for two server programs: an e-mail server program and a World Wide Web server program. Each program operates on data stored exclusively in its own address space.

To protect a program from being corrupted by other programs, a developer can implement portions of an application as a server program. Only the portions of an application that incorporate a human interface need to be implemented in a cooperative program. For example, after a user writes an electronic mail message with an e-mail editing program, that cooperative program can call an e-mail server program and request the server program to deliver the message over a network. Likewise, the e-mail server program can

FIGURE 3.2 Server programs protected by separate address spaces



receive messages sent to the user from across the network and store them until the user is ready to read them with the e-mail editing program.

To protect critical system data and increase system reliability, many non-privileged Mac OS 8 services are implemented as server programs. For example, the Process Manager and the Font Manager (which provides font-rendering services to the system) are implemented as server programs, each in its own protected address space. As you'll see later in this chapter, privileged code—such as the microkernel—has protection mechanisms of its own.

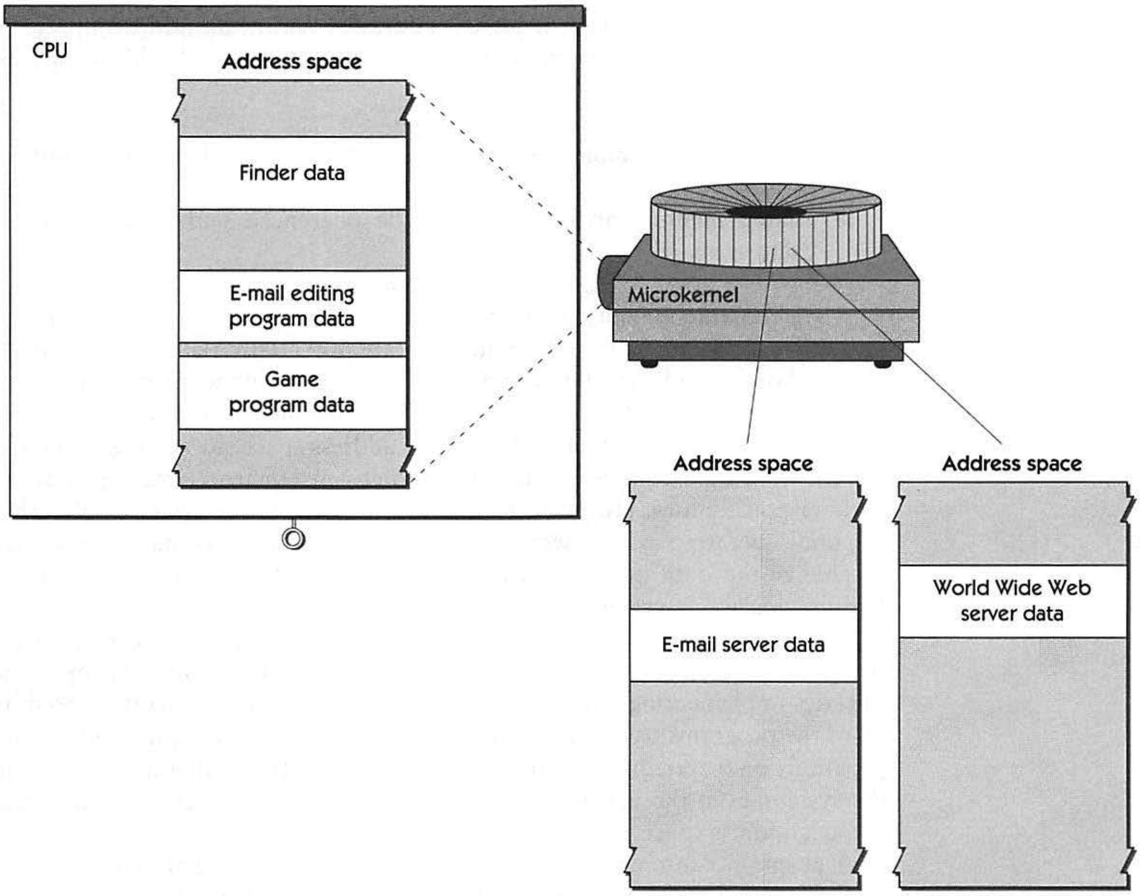
Another benefit to designing software as a server program is that it has an address space all to itself for storing its temporary data. Cooperative programs, by contrast, must share their address space with each other, reducing the amount of address space available to each cooperative program.

ADDRESS SPACE SWITCHING BY THE MICROKERNEL

The CPU can read from and write to the memory of only one address space at a time. The microkernel is responsible for keeping track of all the memory addresses for the code and data residing in these address spaces. The microkernel manages these address spaces so that the CPU works with only one address space at a time.

Figure 3.3 symbolizes how the microkernel manages multiple address spaces. In this figure, address spaces are represented as slides in a slide projector. The microkernel operates like the slide projector—while many address

FIGURE 3.3 Switching between address spaces



spaces are available, the microkernel projects only one at a time onto the CPU. In this figure, the microkernel is projecting the cooperative program address space onto the CPU, represented here as a projection screen. When the microkernel determines that it's time for one of the server programs to execute on the CPU, the microkernel "projects" that program's address space onto the CPU. (Chapter 4 explains how the operating system determines which task of which program gets to execute on the CPU at any given moment.)

System-Wide and Shared Memory Areas

A memory area is a range of logical addresses within an address space. In addition to supporting memory areas specific to individual address spaces, Mac OS 8 also maintains

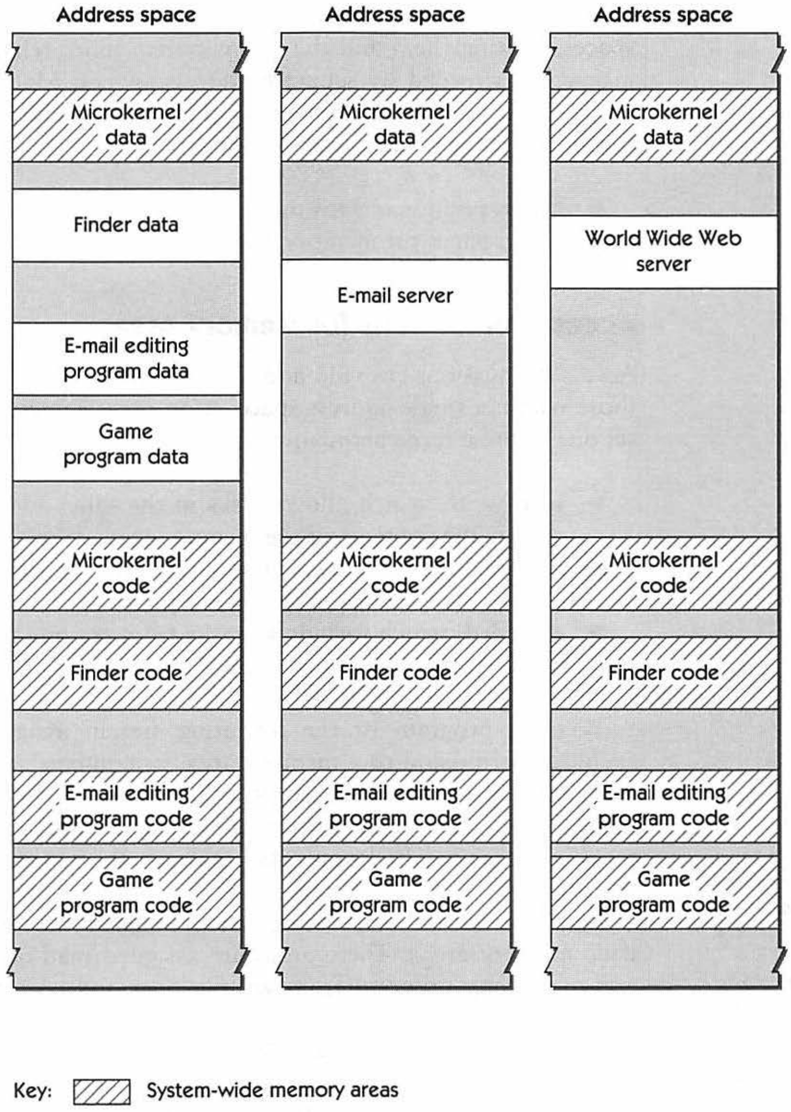
- **system-wide memory** areas, which can be referenced across all address spaces
- **shared memory** areas, which can be referenced within two or more address spaces

A system-wide memory area appears at the same location in every address space. The contents of a system-wide area are potentially visible in all address spaces. For example, the microkernel employs system-wide memory areas for storing its own data, as shown in Figure 3.4. The microkernel is essentially a process that exists simultaneously in every address space. By storing its data in system-wide memory areas, the microkernel can efficiently manage system-wide responsibilities. (To protect the stability of the entire system, only other essential operating system services—such as device drivers—have permission to change the data in the microkernel's system-wide memory areas. Access permissions are described in the next section.)

The operating system also maps all executable code into system-wide memory areas. Thus, a single copy of the code from any library—such as any of the libraries implementing operating system services—can be efficiently shared by all of the programs using that library. As Figure 3.4 illustrates, the code for all programs on a system exists in identical locations across all address spaces in the system, even though the programs store their data in memory areas local to each address space.

A program can create a system-wide memory area to share its data with programs in other address spaces. More likely, however, a program will use a shared memory area for this purpose. A shared memory area exists in two or more address spaces, but not necessarily all address spaces. A shared memory area can begin at the same address in various address spaces (which is useful if shared data is accessed by pointers, because pointers contain memory addresses), or it can begin at different addresses. A shared memory area can have different access permissions in different address spaces. For example, a program can write data into a shared memory area in its own address space but, as you'll see in the next section, make the data read-only to programs in other address spaces, thereby granting other programs access to a reliable copy of the data.

Figure 3.4 System-wide memory areas



ADDITIONAL FORMS OF MEMORY PROTECTION

You've seen how Mac OS 8 separates server programs into their own address spaces, making them and the entire system more reliable. In addition to the protection afforded by separate address spaces, Mac OS 8 offers two more levels of memory protection that reduce the possibility of one program corrupting the code or data used by another:

- ▶ access permissions for memory areas
- ▶ guard pages for memory areas

Access Permissions for Memory Areas

Access permissions provide additional protection to memory areas, even to those within a single address space. A program can create a memory area and set one of these three permission levels:

- ▶ **read/write,** which allows tasks in the same address space to view and change the contents of the memory area
- ▶ **read-only,** which allows tasks in the same address space to view but not change the contents of the memory area
- ▶ **excluded,** which forbids all tasks from reading from and writing to the memory area

When a program or the operating system assigns either read-only or excluded permission to a memory area, its contents are safe from corruption from other programs because no other program can write to that memory area. If a program or the operating system attempts to access a memory area to which it has insufficient access privileges, the processor generates an exception.

An **exception** is an error or other special condition that is detected by the CPU during code execution. An exception transfers control from the code generating the exception to another piece of code, usually an exception handler.

As you've seen, the operating system maps all executable code into system-wide memory areas. These areas are assigned read-only permission, thereby preventing any program from writing over and corrupting the code of any other program.

If a program needs to share data with other programs, it can create a read-only memory area for the data. The creator of a memory area can also specify separate access permissions for nonprivileged and privileged code. **Nonprivileged code** is executed while the CPU is in user mode. **User mode,** in turn, is a state of operation for the PowerPC CPU that protects certain processor resources, such as various processor registers, from being modified. Nonprivileged code is restricted from using various CPU instructions and hardware addresses and from changing data used by critical portions of the operating system. (To protect the stability of the user's system, most code in Mac OS 8 runs while the processor is in user mode.)

A **processor register** is a named area of high-speed memory located on the CPU.

Only the code for device drivers, the microkernel, and some other portions of the operating system is privileged. **Privileged** code is executed while the CPU is in supervisor mode. **Supervisor mode**, in turn, is a state of operation for the PowerPC CPU that allows full access to critical processor resources, such as all processor instructions and the tables that control memory protection. Privileged code can execute CPU instructions that are restricted from nonprivileged code and can access hardware addresses invisible to nonprivileged code.

The data used by privileged code can be excluded from nonprivileged code. A device driver, for example, may create a memory area that allows read/write access to privileged software but read-only access to nonprivileged software. Even privileged software can be denied write access to a memory area. For example, the system-wide memory areas containing code are always assigned read-only access for both privileged and nonprivileged software. Video RAM, which also resides in a system-wide memory area, is assigned read/write permission for both nonprivileged and privileged code.

(As a sidelight, it should be noted that to help protect system reliability, only privileged code can switch the CPU between supervisor mode and user mode. The microkernel always runs in supervisor mode; functions that call the microkernel cause the CPU to switch to supervisor mode. Before returning execution control back to nonprivileged code, the microkernel switches the CPU back to user mode.)

Guard Pages

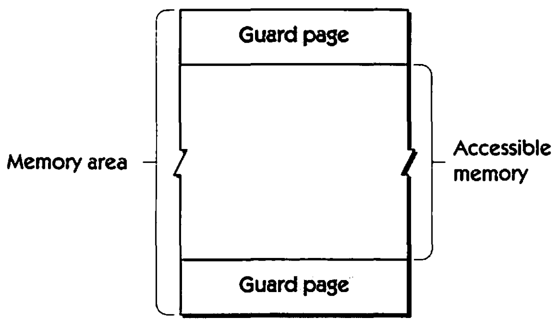
A **page** is the smallest unit, measured in bytes, of information that the virtual memory system can transfer between physical memory and backing store. As you'll see in Chapter 6, a memory area is always a multiple of some number of pages.

Guard pages provide another level of protection, even to memory areas with read/write permission. When any program is launched in Mac OS 8, the operating system automatically places one or more guard pages at each end the program's stack and around the areas (sometimes known as **heaps**) created for its dynamic memory allocation needs. A program can specify its own number of guard pages to appear at the beginning and end of these areas and around any additional memory areas it creates. Mac OS 8 allows no access whatsoever to guard pages; neither privileged nor nonprivileged software can write to or read from them.

A **stack** is a memory area where a task stores some of its temporary variables during execution. A **stack frame** is the area of the stack used by a routine for its parameters, return address, local variables, and temporary storage.

Figure 3.5 illustrates a memory area with guard pages. If any code, even for the program using that memory area, attempts to access a guard page, the CPU generates an exception. For example, a program can surround its stack with a range of guard pages equal to the length of its maximum stack frame. These guard pages then prevent the program's stack from overflowing into the memory area of any other program. If the stack were to overflow and the stack attempted to access one of its guard pages, the CPU would send an exception to the program with the overflowing stack, resulting in the termination of that program before it could adversely affect any adjoining memory areas.

FIGURE 3.5 A memory area with guard pages



SUMMARY

Mac OS 8 uses multiple address spaces. The microkernel manages the system's multiple address spaces so that the CPU always references the right address space at the proper time.

By separating server programs into their own address spaces, Mac OS 8 protects these programs, making them and the whole system more reliable. Cooperative programs share a single address space to support System 7 application compatibility. Within this 4-GB address space, the large amount of addressable memory virtually eliminates memory fragmentation problems so that the user can open the greatest possible number of cooperative programs.

Mac OS 8 provides other forms of memory protection, too. First, programs as well as the operating system can assign read-only or excluded privileges to memory areas, thereby limiting access to and possible corruption of these areas by other programs. The operating system, for example, loads all code in areas that permit read-only access. Second, a program can place guard pages around a memory area to help prevent the program from accidentally accessing adjacent memory areas.

In order for code and data to be shared among address spaces, Mac OS 8 provides system-wide memory areas, which are visible in every address space, and shared memory areas, which are visible only in the address spaces of the programs that need access to these areas.

PLANNING A PRODUCT FOR MAC OS 8

If you're a developer, you can begin preparing to take advantage of multiple address spaces by determining whether some portion of your product benefits from the extra protection afforded by a separate address space. If so, you should plan to implement this portion as a server program.

4

The Architecture of the Multitasking Mac

A multitasking operating system allows multiple programs to execute in a simultaneous or nearly simultaneous manner. Mac OS 8, unlike previous versions of the Mac OS, performs preemptive multitasking—that is, Mac OS 8 actively controls moment-to-moment program access to the CPU. Through the use of a cooperative scheduling policy for cooperative programs, Mac OS 8 also allows System 7 applications to run efficiently in this preemptive multitasking environment.

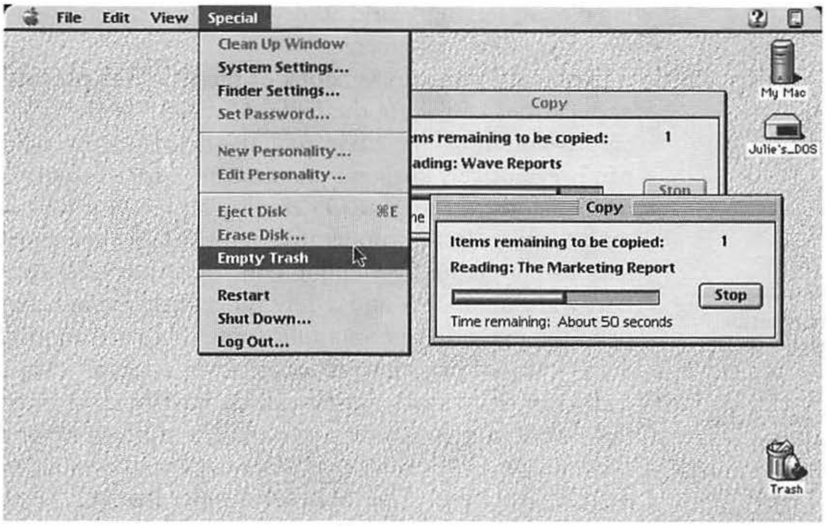
Preemptive multitasking makes efficient use of the computer. For example, the user can continue working even while background operations—such as e-mail transactions, automated file backups, and complex image-rendering calculations—are taking place. This type of system performance translates to improved user productivity.

To the user, it appears that multitasked operations take place simultaneously. A CPU can execute only one operation at a time, but the microkernel interleaves the execution of these operations so quickly that it looks as if they're happening simultaneously. For example, between the time that a user selects a paragraph of text and chooses the Copy command, the microkernel may have instructed a single-CPU system to execute a network I/O operation, perform part of a statistical calculation, or execute several system operations. On a Mac OS 8–compatible computer with more than one processor, multiple tasks like these actually *do* run simultaneously.

KEY TERMS AND CONCEPTS

- ▶ **Preemptive multitasking** is the ability of an operating system to allocate access to the CPU and other operating system services among multiple tasks, thereby allowing multiple programs to execute in a nearly simultaneous manner. The Mac OS 8 microkernel provides the mechanisms for preemptive multitasking.
- ▶ A **task** is the basic unit of program execution in Mac OS 8. A task is always associated with a process, and several tasks can be associated with a single process. Whenever any task within a process is eligible for execution, the microkernel preemptively schedules the task for execution along with all other eligible tasks. Tasks that are temporarily ineligible for execution are said to be **blocked.**
- ▶ A **process** is an instance of a program at execution time. A process is characterized by a set of one or more tasks and the memory and other operating system resources allocated to those tasks. (Mac OS 8 uses processes for tracking and reclaiming these resources.)
- ▶ **Preemptive scheduling** is the policy by which the microkernel allocates moment-to-moment access to the CPU among all eligible tasks. The microkernel uses a set of well-defined rules to schedule which task should execute at any given time. Following these rules, the microkernel can suspend the execution of one task and resume the execution of another. Preemptive scheduling is necessary for preemptive multitasking. To synchronize access to the cooperative services and provide compatibility for System 7 applications in this preemptively scheduled environment, Mac OS 8 also employs a scheduling policy called cooperative scheduling.
- ▶ **Cooperative scheduling** is the Mac OS 8 policy for scheduling access to cooperative services. When programs cooperate by yielding execution eligibility to one another in their event-handling code (described in Chapter 14), the Process Manager serializes their calls to the cooperative services. This serialization allows each call to execute to completion without being interrupted by another call to the same service. Cooperative scheduling rotates eligibility among the main tasks of cooperative programs so that each can, in turn, be preemptively scheduled with all other tasks in the system.
- ▶ A **main task** is the first task created by the microkernel for a process. The main tasks for cooperative programs can safely use Mac OS 8 cooperative services, whereas all other tasks in Mac OS 8 must use only reentrant services. A developer can design a program so that after it becomes instantiated as a process, it may contain other tasks in addition to its main task.

FIGURE 4.1 Multitasked operations in Mac OS 8



ms remaining to be copied: 1
ading: Wave Reports

Items remaining to be copied: 1
Reading: The Marketing Report
Time remaining: About 50 seconds

MAJOR POINTS OF INTEREST

To let the user make the most productive use of the computer, the microkernel will preempt the execution of one task and start or resume the execution of a more urgent task. For example, the microkernel might suspend the execution of a file compression program to let a multimedia program prepare video data for immediate onscreen play. At the next opportune moment, the microkernel resumes execution of the file compression operation.

Preemptive multitasking also makes efficient use of the CPU by keeping it as busy as possible. For example, because the CPU executes instructions faster than I/O devices transfer data, the microkernel will suspend the execution of a program that is waiting for data to come off a disk. The operating system then schedules other operations for execution, keeping the CPU busy even while the program is waiting for data from the disk. When that data becomes available, the microkernel reschedules the program for execution.

Even while the microkernel continually suspends and resumes task execution to keep the CPU busy, the operating system remains highly responsive to users. Figure 4.1 illustrates multiple tasks operating concurrently. The user has begun copying one large file to a computer connected on a network and then begun copying a second large file to another computer. As the CD-ROM animated version of this figure demonstrates, before either copy operation is finished, the user can delete files and launch and interact with multiple applications. At the same time that all of this activity takes place, the CPU can also execute other operations that are invisible to the user—operations such as receiving and storing e-mail, serving World Wide Web pages to remote users, and compressing or decompressing large files.

Although previous versions of the Mac OS provide mechanisms by which applications can share the CPU, Mac OS 8 is the first Mac OS to offer a preemptive multitasking environment—that is, one in which the operating system can preemptively suspend one task in order to allow others to execute. Previous versions of the Mac OS employ a type of multitasking called **cooperative multitasking.** In the cooperative multitasking environment of System 7, applications cooperate by yielding control of the CPU to one another. Except for hardware interrupts and a few other mechanisms that can momentarily suspend the execution of an application, there are no mechanisms in that operating system for preemptively controlling application access to the CPU.

A **hardware interrupt** is an exception signaled to the CPU by a hardware device, notifying the CPU of a change of condition in the device. In response, the CPU momentarily suspends application execution to execute a routine called an interrupt handler.

In the cooperative multitasking environment, developers must program task scheduling into their software. It's impossible for developers to anticipate the scheduling priorities that will exist on any system when their programs are actually running. The Mac OS 8 microkernel, however, does know from moment to moment what demands are being made of the computer, and the microkernel preemptively schedules task execution according to the priority of these demands.

As you read in Chapter 2, the cooperative services aren't reentrant. Therefore, Mac OS 8 uses a policy called **cooperative scheduling** to serialize access to these services. System 7 applications and OpenDoc parts are automatically compatible with this cooperative scheduling policy. Cooperative scheduling allows only one task at a time to be eligible to call the cooperative services, thus preventing one program from preempting another program's call to such a service. The single eligible task in the cooperative scheduling environment is preemptively scheduled with all other eligible tasks across the system, such as tasks for server programs and for the operating system itself.

As you'll learn in the next chapter, developers can design multiple threads of execution into their software products to increase system efficiency and user responsiveness. A multithreaded program is structured into parallel operations, each of which gets access to the CPU. This feature allows a user to continue working within an application without waiting for the application to complete lengthy operations. For example, a scientific simulation application could create one task that handles user interaction and another task that performs intensive statistical calculations in the background. The user can continue to interact with the program even while it's performing statistical calculations.

The microkernel interleaves the execution of multitasked operations so quickly that it looks as if they're happening simultaneously on computer with one CPU. On a Mac OS 8–compatible computer with more than one processor, however, multiple tasks actually *do* run simultaneously. Developers don't need to perform any special programming to take advantage of multiprocessor computers; instead, the microkernel automatically schedules tasks to run on all available processors.

MAC OS HERITAGE

Cooperative Multitasking in System 7 and System 6

Unlike Mac OS 8, where the microkernel preemptively controls access to the CPU, System 7 employs a cooperative multitasking environment, where the currently executing application exclusively decides when to relinquish access to the CPU. Typically, a System 7 application processes events relating to user and computer activity; when there are no events for the application to handle, the application yields and lets other applications execute. In this way, applications take turn handling events and sharing execution time on the CPU.

In the first Macintosh Operating System, the computer was designed to be used with only one application at a time. In this environment, preemptive scheduling offered very little benefit, and so reentrancy wasn't built into the operating system. As the Macintosh computer grew in capabilities, it became feasible for users to work with several programs simultaneously. To allow users to open multiple programs while continuing to run older applications, the System 6 MultiFinder introduced an environment where applications cooperated to share access to the nonreentrant services of the operating system. System 7, in turn, fully incorporated this environment, providing backward compatibility for applications developed for System 6.

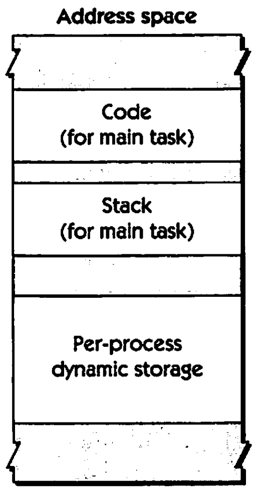
PROCESSES AND TASKS

Mac OS 8 **processes** are analogous to processes in the UNIX, Windows NT, and Windows 95 operating systems. Mac OS 8 **tasks** are analogous to the **threads** in these systems. Apple uses the term *task* to avoid confusion with the threads created by System 7 developers using the Thread Manager. See the next chapter for more information about threads in Mac OS 8.

In Mac OS 8, a task is the basic unit of program execution; it performs a sequence of programmatically defined operations. When launched, every program has at least one task, called its main task. The main task of a video-effects editing program, for example, might present an interface that allows the user to select from or create a variety of transitional effects when sequencing video tracks. A process may incorporate other tasks as well; for a video-effects editing program, a second task may be employed to write and read video data to and from a hard disk.

In Mac OS 8 nomenclature, a process is a passive entity—that is, a process isn't executable. Instead, it is the tasks associated with a process that are executable. In order to be executed, every task requires various operating system resources, such as a set of processor registers and memory areas for storing its temporary data. All of the resources allocated to a task are packaged by the microkernel into a process, which the operating system uses to track these resources.

Figure 4.2 Memory areas for a Mac OS 8 process with one task



When a program is launched—for instance, when a user double-clicks its icon—the operating system prepares the program code for execution, creates memory areas for the code and its temporary data, and assigns locations for the code and data within these memory areas. In this way, the program becomes instantiated as a process. Figure 4.2 illustrates some of the memory resources required by a process with a single task:

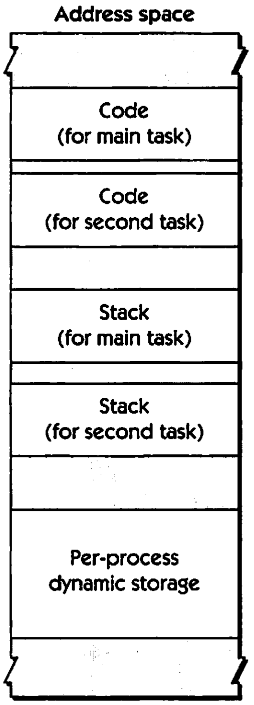
- ▶ a memory area for the executable code of the task itself
- ▶ a stack for use by the task
- ▶ one or more memory areas for dynamic storage allocation by the task

A **stack** is a memory area where a task stores some of its temporary variables during execution. The Dynamic Storage-Allocation Service, described in **Chapter 7,** supplies additional memory areas from which a program can dynamically allocate and release memory storage for temporary data during execution.

Every task gets its own stack and set of CPU registers. However, as Figure 4.3 shows, if a process contains more than one task, all tasks in the process initially share the same dynamic storage memory area, in which the tasks save such temporary data as data structures and global variables. Mac OS 8 automatically supplies additional memory areas if they become needed by a program during execution time, and programs can allocate additional memory areas for their dynamic storage.

When a cooperative program is launched, the operating system instantiates a process for it within the address space shared by all cooperative programs. When a server program is launched (usually this happens automatically when the user starts the computer), the operating system creates a new address space and instantiates a process for the server program within that address space. When a process is terminated (as when a user quits an application), the oper-

FIGURE 4.3 Memory areas for a Mac OS 8 process with two tasks

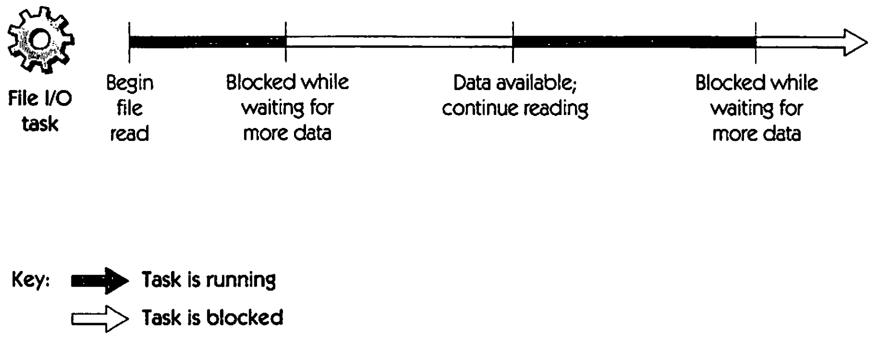


ating system releases all of the resources related to that process—an improvement over System 7, which leaves various system resources in memory if the application using them is abnormally terminated, such as when a programming error within an application causes it to crash.

TASK SCHEDULING

After creating a process, the operating system schedules its tasks for execution. At any moment, the main task of only one cooperative program can be eligible for execution, thereby synchronizing access to the cooperative services. This and all other eligible tasks are then scheduled preemptively by the microkernel according to its priority-based scheduling algorithms.

FIGURE 4.4 A task blocking on a synchronous I/O operation


File I/O task
Begin file read
Blocked while waiting for more data
Data available; continue reading
Blocked while waiting for more data
Key:
Task is running
Task is blocked

Preemptive Scheduling

The microkernel preemptively schedules moment-to-moment access to the CPU among all eligible tasks. When it determines that an eligible task should execute, the microkernel gives it access to the CPU by suspending the currently executing task. This transition point, where the currently executing task is suspended and the execution of a different task is undertaken, is called a **context switch**. During a context switch, the microkernel saves the execution state of the suspended task and replaces it with the execution state of the task about to execute.

Eligibility

Not all tasks are eligible for execution. Tasks that aren't eligible for execution are said to be blocked on some condition, such as the completion of a synchronous I/O operation. When a task is blocked on some event and that event occurs, the task becomes eligible for execution again.

Figure 4.4 illustrates how a task performing a synchronous file read operation is blocked whenever the task is forced to wait for data from the disk drive. The task becomes eligible for execution as soon as the disk drive has sent more data. Whenever the task in this figure is blocked, the microkernel sends other tasks to the CPU for execution.

A **synchronous I/O operation** is one where the task requesting input to or output from a device cannot continue executing until the operation is finished.

Scheduling Policies

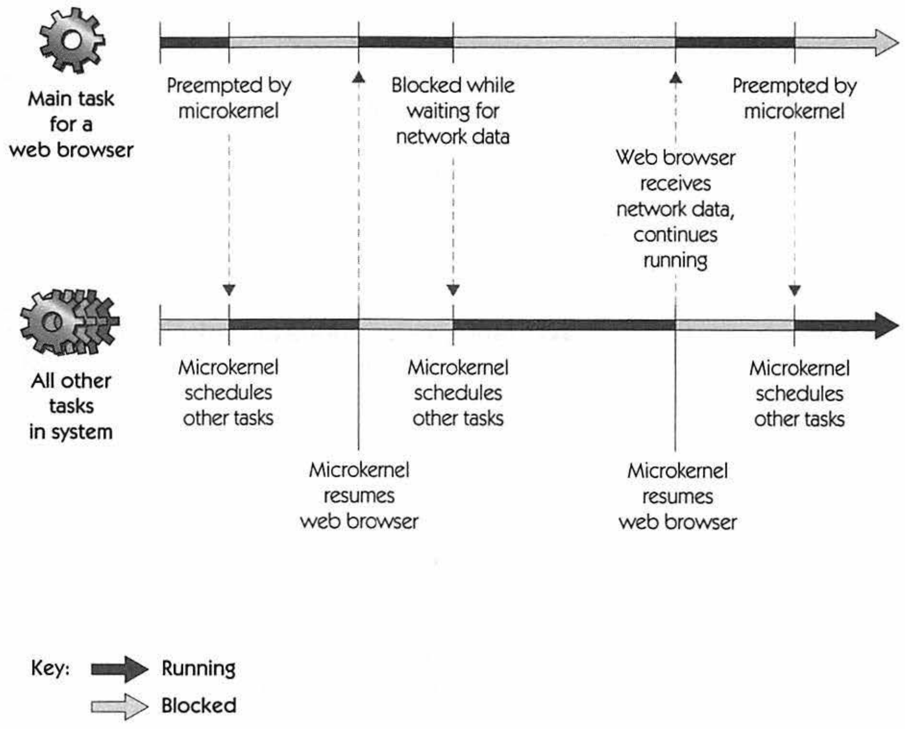
Many tasks can be eligible for execution, but the CPU can execute only one task at a time. The microkernel determines which task gets to be executed. The first criterion for making this determination is priority: the task with the highest priority is given execution precedence.

Developers assign priorities to their program tasks. The following list shows the general priority levels for tasks and the types of programs to which these priorities are typically assigned.

- Real-time tasks, such as those for sound playback and video capture. These tasks are given the highest priority. Because of the time-critical nature of a real-time task, it executes until it's blocked, and it immediately resumes execution as soon as the task becomes eligible again. Very few programs employ tasks assigned this level of priority.
- Operating system tasks, such as those for the I/O system and the Process Manager. Only real-time tasks have higher priority. Because operating system tasks generally execute for such short durations, the CPU spends very little time executing the tasks at this priority level.
- Server program tasks. The CPU spends much of its time executing tasks at this priority level.
- The main tasks of cooperative programs. As you'll read in the next section, no more than one is ever eligible for execution.
- Additional tasks created for cooperative programs. Developers generally assign these tasks a lower priority than they assign to their main tasks. However, a developer may assign a higher priority, such as the priority for a server program, to a cooperative program's additional tasks. For example, if a cooperative program uses an additional task to perform critical data I/O, the additional task may be assigned a higher priority than the main task. Such a task would spend most of its time waiting for I/O operations to complete, so it wouldn't degrade the user responsiveness of the cooperative program's main task, yet the additional task's higher priority would ensure that it quickly gets the small amount of execution time needed to initiate the next I/O operation.
- Lowest priority tasks. These tasks generally get to execute only when there are no other eligible tasks. For example, a task with this priority might be used to perform automated file backups when the user is away from the computer.

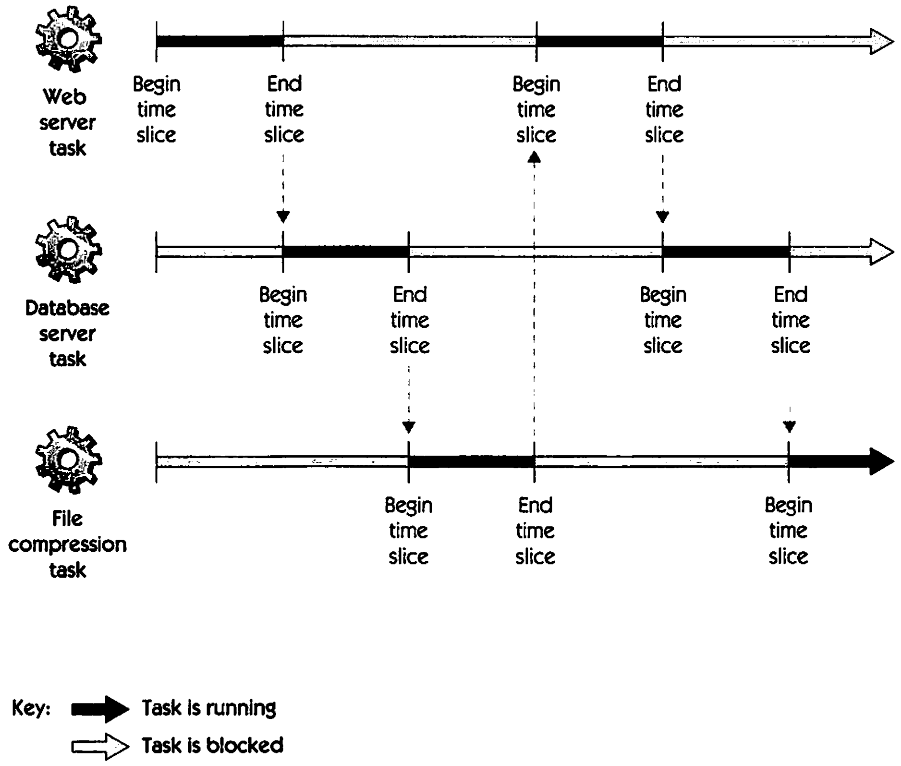
A task with one of the highest levels of priority—that is, a real-time task or a system task—always executes until it's blocked. If there are no tasks eligible at these priority levels, the task with the next highest priority on the system is given precedence to execute on the CPU. At the time this book went into production, Apple Computer was tuning its scheduling algorithms so that the highest-priority task is given access to the CPU somewhat less than 100 percent of the time (unless the task's a real-time or system task, both of which always get full access to the CPU). This scheduling approach prevents a task from starving lower-priority tasks from execution. Apple intends to refine these algorithms based on the performance of programs running on prerelease versions of Mac OS 8.

FIGURE 4.5 Preemptive task scheduling by the microkernel


Main task for a web browser
Preempted by microkernel
Blocked while waiting for network data
Web browser receives network data, continues running
Preempted by microkernel
All other tasks in system
Microkernel schedules other tasks
Microkernel schedules other tasks
Microkernel schedules other tasks
Microkernel resumes web browser
Microkernel resumes web browser
Key:
Running
Blocked

According to their priorities, then, the microkernel shuffles eligible tasks on and off the CPU. This is illustrated in Figure 4.5, where CPU time for the main task of a World Wide Web browser program is indicated by the shaded portions of the time line at the top of the figure. The CPU time allotted to all other tasks in the system is illustrated by the shaded portions of the bottom time line. At the beginning of the time represented in this figure, the browser's main task has the highest priority and hence executes. As soon as a task with a higher priority becomes eligible, the microkernel preempts the browser's task and performs a context switch to allow the higher-priority task to run. When the main task of the browser has the highest priority again, the microkernel performs another context switch and resumes execution of the browser's main task. If this task becomes blocked waiting for the arrival of data from the network, the microkernel performs another context switch and allows the task with the next highest priority to execute. When the network data becomes available to the browser's main task, the task becomes eligible, and the microkernel schedules it once more for execution.

FIGURE 4.6 Execution time divided by time slices for tasks with equal priority



Time Slicing

A task with one of the highest levels of priority—that is, a real-time task or a system task—always executes until it's blocked, even if another task with equal priority becomes eligible. However, for tasks at most other priority levels, the microkernel uses a type of scheduling called **time slicing** to allow tasks of equal priority to share CPU time. In this scheduling policy, when multiple tasks have the same priority, and that becomes the highest priority on the system, the microkernel allows each task to execute for an internally specified time interval called a **time slice.** When a time slice expires, the microkernel switches to the next task with the same priority.

Figure 4.6 illustrates tasks for three server programs. Because they're all eligible and all share the same priority level, the microkernel divides execution time among them so that each task executes for a time slice before the microkernel switches to another task.

Cooperative Scheduling

The Process Manager coordinates scheduling for the main tasks of all cooperative programs. From among all such main tasks on the system, no more than one at any moment can be eligible for execution. The microkernel, in turn, preemptively schedules this task for execution. By rotating eligibility among the main tasks of all cooperative programs, the Process Manager gives each the opportunity to execute.

Synchronizing Access to the Cooperative Services

As described in Chapter 2, the cooperative services support the Mac OS 8 human interface and maintain compatibility with applications written for System 7. Because they aren't reentrant, cooperative services must complete the request from one task before receiving another. Otherwise, data could be corrupted were one task to preempt another. For example, suppose one program has called the Human Interface Toolbox to create a window. The Human Interface Toolbox must finish creating that window before receiving any more requests to create windows; otherwise, window-creation operations will fail.

To synchronize calls to its cooperative services, Mac OS 8 defines an environment for cooperative scheduling, so called because programs cooperate to safely schedule access to these services. The main thread of a cooperative program

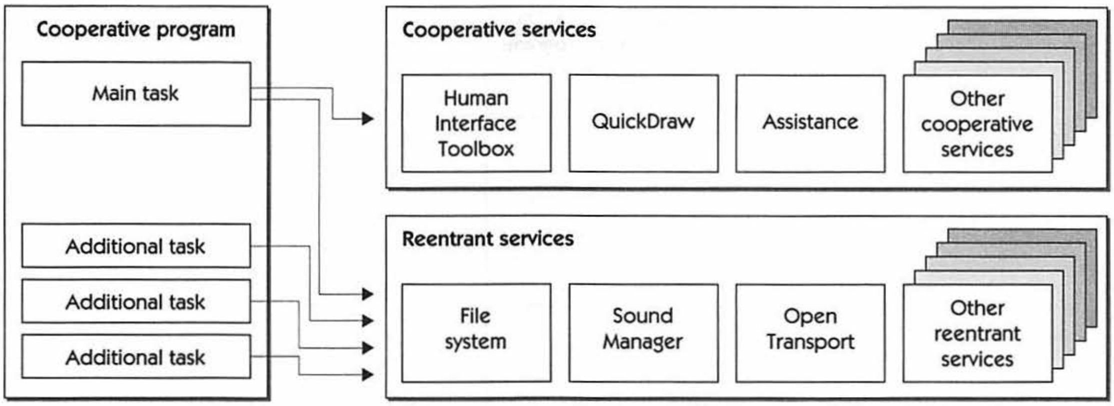
- yields eligibility whenever there are no events for it to respond to
- contains all of the program's calls to the cooperative services

When a cooperative program is launched—for instance, when a user double-clicks its icon—the Process Manager instantiates a process for that program within the address space shared by all other cooperative programs, and the operating system creates a main task for the newly instantiated process. The main task then waits for the operating system to send it **events**—user actions or system occurrences to which it must respond. Events include keystrokes and mouse clicks from the user, requests from other programs (for example, to print files), or any other activities in the system (for example, the completion of I/O operations).

When there are no events for a main task to handle, it yields its eligibility to execute. It becomes eligible for execution as soon as an event for it arrives. When the microkernel gives that task access to the CPU, the task responds to the event and then yields its eligibility again, thereby allowing other tasks on the system to efficiently share the CPU.

An OpenDoc part doesn't have a main task. Instead, OpenDoc creates a main task for every OpenDoc document when it's opened by the user. For every OpenDoc document, all part editors run within the main task of that

FIGURE 4.7 Making all calls to cooperative services from the main task of a cooperative program



document. The OpenDoc environment automatically ensures safe access to the cooperative services from OpenDoc parts.

As you'll see in the next chapter, a developer can incorporate multiple threads of execution within a cooperative program by creating additional tasks. OpenDoc developers can also create additional tasks for their OpenDoc parts. As Figure 4.7 shows, the cooperative services are called exclusively from a cooperative program's main task. The main task and all additional tasks can call the reentrant services.

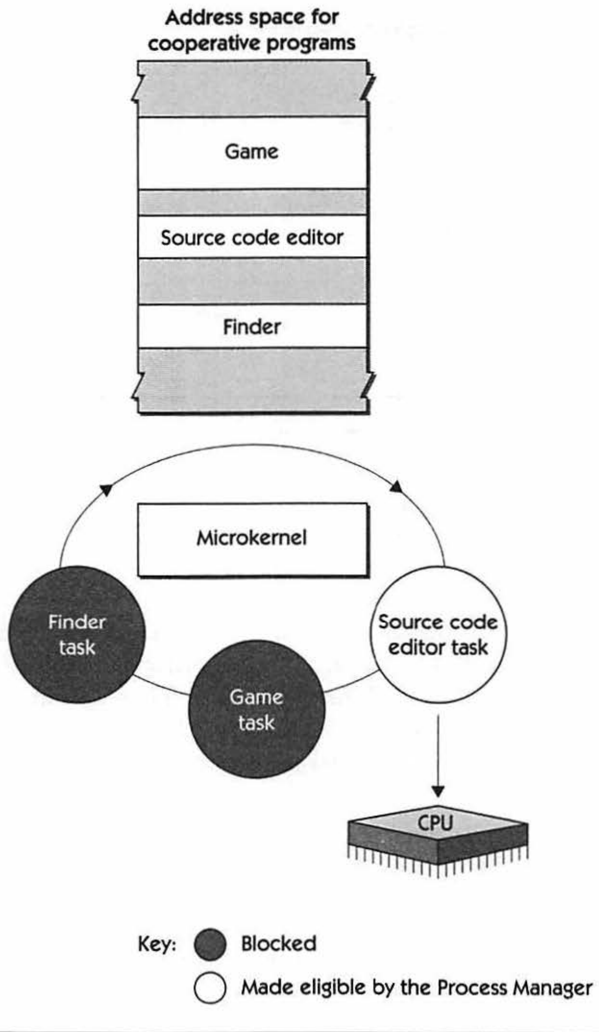
Whereas the microkernel preemptively schedules the execution of all eligible tasks, only one cooperative program can be eligible for execution at a time. For example, the moment the user interacts with an e-mail application, such as by moving the cursor to the application's menu bar and pressing the mouse button, the Process Manager makes the main task of the e-mail application eligible for execution. The main tasks of all other cooperative programs are blocked. With only one cooperative program eligible for execution at a time, the Process Manager serializes all calls to the cooperative services.

The Blocking of Main Tasks

Whenever there are no events for a cooperative program to respond to, the main task for that program yields its eligibility and becomes blocked. (Program event handling is described in Chapter 14.) If no user or system events are pending for any cooperative programs, *all* of their main tasks may become blocked.

Figure 4.8 illustrates three applications operating in the cooperative program address space: a game, a source code editor, and the Finder. The process for each cooperative program has a main task, and of these three, only the

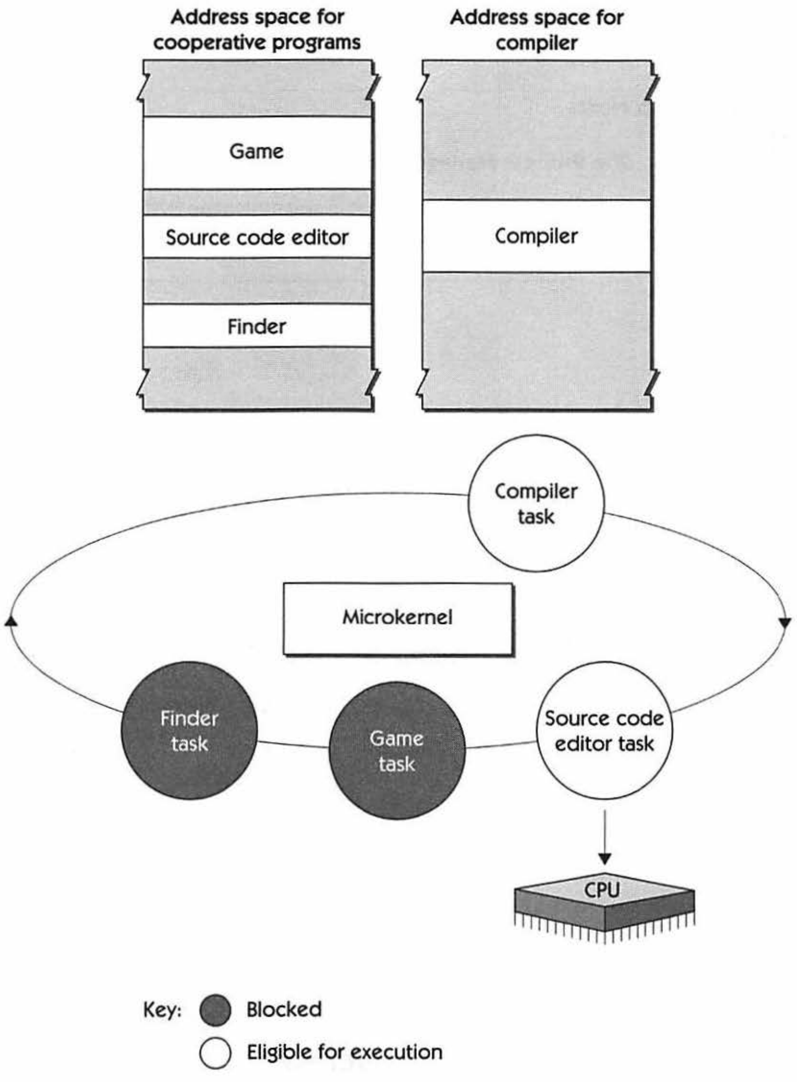
FIGURE 4.8 Cooperative scheduling of main tasks for cooperative programs



main task for the source code editor is eligible for execution. The other main tasks have yielded and are blocked. In this figure, the microkernel grants the main task for the source code editor access to the CPU for execution.

Figure 4.9 shows a server program that compiles code created with the source code editor. Outside of the the cooperative scheduling environment, any number of tasks may be simultaneously eligible for execution. Although many tasks may be eligible on the system, a CPU can execute only one at a

FIGURE 4.9 Preemptively scheduled tasks for a cooperative program and a server program



time. The microkernel uses its priority-based preemptive scheduling rules to decide which task that will be.

In this example, the source code editor has been assigned a higher priority than the compiler, so the microkernel allows the source code editor to continue to execute. However, if the source code editor were to become blocked along with the other cooperative programs (as if, say, all were waiting for events), then the compiler would begin executing. Or if another task with a higher priority were to become eligible, the microkernel would preempt the source code editor and allow the higher-priority task to execute.

COMPATIBILITY NOTES

The Process Manager

The Mac OS 8 Process Manager subsumes the System 7 Process Manager. Mac OS 8 supports all of the functions and data structures of the Process Manager from System 7.

SUMMARY

For increased system efficiency, Mac OS 8 performs preemptive multitasking. This allows the CPU to remain as busy as possible. The microkernel uses priority-based scheduling algorithms to determine which task gets immediate access to the CPU: the more urgent the task, the sooner it gets execution time.

Not all tasks are eligible for execution. For example, a task waiting for a file I/O operation to complete will become blocked so that the CPU can immediately begin executing some other task. When the file I/O operation completes, the previously blocked I/O task becomes eligible to execute again.

Cooperative programs yield execution eligibility to one another in the event-handling code of their main tasks. This cooperative yielding of control allows the Process Manager to serialize all calls to the cooperative services. Every call to one of these services executes to completion without being interrupted by another call to the same service. Through this arrangement, the main tasks of cooperative programs are able to safely call the cooperative services. All other tasks must use only the reentrant services.

PLANNING A PRODUCT FOR MAC OS 8

If you're an application developer, you should consider whether any of your application code can be implemented in a task that's separate from your application's human interface code. Such an implementation could take fuller advantage of preemptive multitasking and increase overall system efficiency.

5 Multithreaded Programs

Much like the operating system makes the most efficient use of computer resources through its multitasking capabilities, a program can make the most efficient use of computer resources by incorporating multithreading capabilities. Whereas multitasking efficiently interleaves the execution of multiple programs on a single CPU, multithreading efficiently interleaves multiple paths of execution within a single program or set of programs. For example, one thread of execution in a program might handle user interactions, another might perform calculations, and a third might perform file I/O.

Multithreading makes an application highly responsive to the user while increasing overall system performance. On multiprocessor computers, where tasks execute simultaneously on multiple processors, multithreading can offer significant performance gains to high-end types of applications.

Mac OS 8 developers can thread products using one or a combination of three different approaches. Developers can divide operations so that they're performed by

- ▶ more than one task in a single process; for example, by incorporating a main task and one or more additional tasks in a cooperative program
- ▶ tasks in more than one process; for example, by incorporating a user interface task in a cooperative program and background processing tasks in a separate server program
- ▶ more than one cooperatively scheduled thread within a task

KEY TERMS AND CONCEPTS

- A **thread** is path of execution for an application. To thread an application is to give it more than one path of execution.
- A **task** is the basic unit of program execution. Preemptively scheduled and assigned a priority by the microkernel, every task has its own stack and set of registers. The microkernel uses processes to track the resources required by tasks so that every process is associated with at least one task, and several tasks can be associated with a single process.
- A **cooperatively scheduled thread** is one of multiple paths of execution in a task. Within a task, these threads cooperate by yielding execution control to one another. Cooperatively scheduled threads can be scheduled for execution only when the task containing them is running. Although the microkernel preemptively schedules all eligible tasks for execution, programs have execution control over the cooperatively scheduled threads they create. From the main task of a cooperative program, any cooperatively scheduled thread may call the cooperative services.
- On a **multiprocessor computer**, a multithreaded program can send its tasks to separate processors for simultaneous execution. (Whereas a program with multiple tasks can send its tasks to different processors, the cooperatively scheduled threads within a single task all execute on the same processor.)
- Programs use **interprocess communication** to exchange information among tasks within processes or between tasks in different processes.

Note here that Mac OS 8 uses the term *task* to mean the entity that some other operating systems, such as UNIX and Windows NT, refer to as a *thread*. Mac OS 8 usage of *thread* is more abstract because, as listed above, there are three ways to implement multiple paths of execution in Mac OS 8.

Other multitasking operating systems typically allow developers to create multiple *threads* within a process in the way that Mac OS 8 developers can create multiple *tasks* within a process. Mac OS 8 also supports another level of threading—the creation of cooperatively scheduled threads within a task. This level of threading was introduced with the System 7.5 Thread Manager. To avoid confusion with the threads that developers create in System 7.5 using the Thread Manager, Apple has adopted the term *task* to refer to a preemptively scheduled path of execution in Mac OS 8. This book uses *Cooperative Thread Manager* to refer to the Mac OS 8 version of the System 7.5 Thread Manager and uses *cooperatively scheduled threads* to refer to the Mac OS 8 version of System 7.5 Thread Manager threads.

MAJOR POINTS OF INTEREST

Developers can thread their software products to increase system efficiency and user responsiveness. A multithreaded program is structured into parallel operations, each of which gets access to the CPU. This feature allows a user to continue working within an application without waiting for the application to complete lengthy operations. For example, a scientific simulation application could create one thread that handles user interaction and another thread that performs intensive statistical calculations in the background. The user can continue to interact with the program even while it's performing statistical calculations.

To the user, it appears that multithreaded operations take place simultaneously. However, the microkernel interleaves the execution of these operations on a single CPU so quickly that it looks as if they're happening simultaneously. On a Mac OS 8–compatible computer with more than one processor, however, multiple tasks actually *do* run simultaneously. Developers don't need to perform any special programming to take advantage of multiprocessor computers; instead, the microkernel automatically schedules program tasks to run on all available processors.

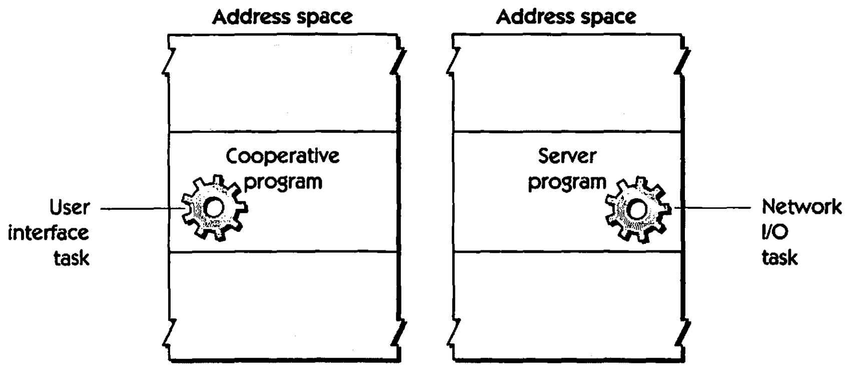
THREADING

Mac OS 8 developers can thread products by using one or a combination of three different approaches. Developers can divide operations so that they're performed

- by more than one task in a single process
- by tasks in more than one process
- by more than one cooperatively scheduled thread within a single task

Not all programs benefit from being multithreaded. For instance, an application that handles user interactions and little else wouldn't benefit greatly by having more than one task. However, if an application presents a user interface while performing time-consuming I/O or processing-intensive operations in the background, its developer provides real user benefit by threading that application.

FIGURE 5.1 Tasks in two separate processes



MAC OS HERITAGE

Threads in System 7.5

The Thread Manager was introduced in System 7.5, where there is no microkernel to perform preemptive task scheduling. For System 7.5 applications compiled to run on PowerPC-based computers, the Thread Manager cooperatively schedules threads for execution. Mac OS 8 likewise cooperatively schedules these types of threads.

Tasks in Different Processes

As you've read in previous chapters, an application may consist of a cooperative program that presents a human interface and a server program that performs I/O or calculation-intensive operations in the background. For example, a World Wide Web application may consist of a cooperative program (with which the user creates and maintains a local web site) and a server program (which makes web pages available to remote users). These two paths of execution make the application multithreaded.

Figure 5.1 illustrates the process for a cooperative program in one address space and the process for a server program in another address space. The main task of the cooperative program manages the user interface for the web-site management portion of the application. The main task of the server program manages network I/O for the web-page server portion of this application.

It's often necessary for these two tasks to communicate information. For example, the server program's task may send network activity information to the cooperative program's task, allowing the cooperative program to display this information to the user. To share this data safely, these tasks must synchronize their access to it. For example, the server program can place this data in a read-only area of shared memory, thereby allowing the cooperative program to read but not change it. As you'll see later in this chapter, Mac OS 8 provides interprocess communication mechanisms for exchanging data between tasks and offers synchronization mechanisms for protecting the data exchanged between tasks.

A developer might choose to thread an application by using separate cooperative and server programs whenever the developer needs one of several tasks to

- be protected within its own address space
- be available whenever the computer is on
- extend its services to multiple client programs

A developer can use a server program to create a task characterized by address space protection, availability, and client extensibility. A developer can then use a cooperative program to create a task that manages user interaction on behalf of the server program.

Address Space Protection

A developer might implement a thread of execution in a server program to protect the operations of that portion of the application. As you may recall from Chapter 3, the operating system builds a separate, protected address space for a server program; therefore, errors in other programs can't corrupt the data of that server program. For example, a database server that supplies information important to the minute-by-minute operations of a company can be implemented as a server program. Thus, even if other programs were to crash on the computer, the server program could continue serving data.

The developer could then implement another thread of execution in a cooperative program, allowing the user to enter, change, and analyze data maintained by the server program.

Availability

A developer might implement a thread of execution in a server program whenever a task should run the entire time a computer is on. Whereas users typically launch and quit cooperative programs, server programs are usually launched automatically when the user starts a computer and run until the user shuts the computer off. For example, a task that receives e-mail messages whenever the computer is on should run as part of a server program. The developer could then implement another thread of execution in a cooperative program, allowing the user to read and respond to incoming messages.

Client Extensibility

When a single thread can be used by multiple other programs, that thread can be implemented as a task in a server program. For instance, a server program that monitors network activity might be useful to a suite of network-management applications, such as an e-mail gateway program, a web-site management program, and a program that backs up remote disk drives.

Other Indirect Interaction Between a Server Program and the User

A task within a server program may also interact with the user indirectly by calling the Notification Manager, a reentrant service, to send a user notification. A **user notification** is an audible or visible indication to the user that a program requires the user's attention. User notifications can take such forms as sounds, icons that flash at the top of the screen, and onscreen alert boxes containing short messages. For example, an e-mail server program can use the Notification Manager to play a sound and display a blinking icon in the menu bar to notify the user of incoming mail. This serves to alert the user of the need to open a cooperative program to read and respond to incoming mail.

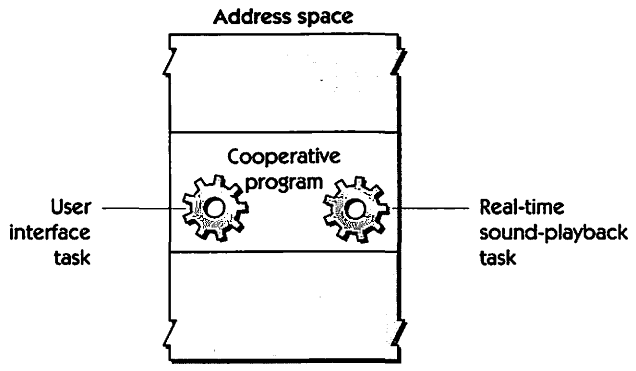
Multiple Tasks in the Same Process

Although threading an application with separate cooperative and server programs has its advantages, the simplest and most straightforward way to thread an application is to create additional tasks in a cooperative program. From its main task, a cooperative program can start additional tasks to offload work that doesn't involve the user interface. By letting the main task manage the program's human interface and using other tasks to perform time-consuming data processing or I/O operations in the background, an application can perform more efficiently and offer greater productivity to the user; by placing multimedia operations in a separate real-time task, an application can perform time-critical multimedia operations without interruption. For example, a multimedia authoring program can use the main task to interact with the user and another task to capture and save video data in real time or to perform real-time sound playback. Figure 5.2 illustrates a cooperative program that incorporates two tasks.

Whereas the main task is cooperatively scheduled against the main tasks of all other cooperative programs in the system, all additional tasks created within a cooperative program are preemptively scheduled with all other eligible tasks in the system. For this reason, time-intensive I/O, computationally intensive data processing, and critical real-time operations make the best use of the CPU when they're placed in tasks other than the main task.

Because the part editors within an OpenDoc document handle user interaction, they use cooperative services. For an OpenDoc document, the main task incorporates the document's constituent part editors, and this main task is

FIGURE 5.2 Two tasks within a cooperative program



cooperatively scheduled like the main task for any other cooperative program. However, a developer can also create additional tasks from an OpenDoc part and, like any additional tasks created for a cooperative program, these tasks must use only the reentrant services.

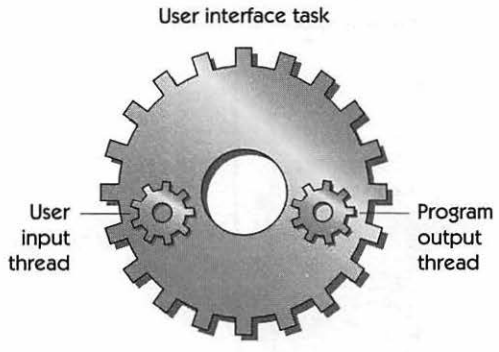
It's often necessary for tasks within the same process to share information. To ensure the integrity of this information, tasks should synchronize their access to it. Compared to data sharing between tasks in separate programs, data sharing between tasks in the same program requires less overhead. As you'll see later in this chapter, Mac OS 8 provides interprocess communication mechanisms for exchanging data between tasks, and Mac OS 8 offers synchronization mechanisms for protecting the data exchanged between tasks.

Cooperatively Scheduled Threads in the Same Task

When it's useful to have multiple paths of execution within a single task, developers use cooperatively scheduled threads. Cooperatively scheduled threads, created with the programming interface defined by the Cooperative Thread Manager, are invoked from tasks. Cooperatively scheduled threads can be scheduled for execution only when the task that created them is running. These types of threads are said to be cooperative because they yield control to one another at programmatically defined times. This prevents one cooperatively scheduled thread from being preempted by any other thread within the same task.

The main task of a cooperative program may call the cooperative services from any of its cooperatively scheduled threads. As illustrated in Figure 5.3, a cooperative program such as a laboratory control application can use one cooperatively scheduled thread to present a window for user input and another thread to present a window for program output. A user of this pro-

FIGURE 5.3 A task with cooperatively scheduled threads



gram could then view data returned from laboratory instruments in one window while controlling the instruments from another window.

Cooperatively scheduled threads can call cooperative services *only* from the main task of a cooperative program. Even when implemented in background tasks, cooperatively scheduled threads are useful for developers who

- want explicit control over when threads yield execution control to one another
- desire simplified synchronization to shared data
- want the switching efficiency of cooperatively scheduled threads

Program Control Over Thread Execution

Unlike preemptively scheduled tasks, which the microkernel can interrupt at any time and cause to execute in any order, cooperatively scheduled threads are executed under the control of the program using them. Within a program, in other words, one cooperatively scheduled thread cannot interrupt another thread. Instead, cooperatively scheduled threads explicitly yield control to each another at points defined by the developer.

Simplified Synchronization

Cooperatively scheduled threads simplify data synchronization. Whereas multiple tasks sharing data must use synchronization mechanisms to access that data safely, a developer doesn't need to employ these mechanisms when using cooperatively scheduled threads. Because a task's cooperatively scheduled threads cannot preempt one another, access to the data they share is automatically serialized. A developer simply needs to ensure that each cooperatively scheduled thread yields control only *after* making changes to data shared with another cooperatively scheduled thread.

Efficient Switching

Because cooperatively scheduled threads are called from a task, the microkernel doesn't perform a context switch whenever a task switches between them. A task is therefore slightly more efficient at switching between cooperatively scheduled threads than the microkernel is at switching between preemptively scheduled tasks.

Multiprocessor Support

On multiprocessor computers, Mac OS 8 executes multiple tasks simultaneously. A **multiprocessor computer** has more than one processor to execute instructions. For example, the Genesis MP computer from DayStar Digital is a Mac OS–compatible computer that includes four PowerPC processors for simultaneously running parallel threads of execution. The Genesis MP platform is an example of an **asymmetric multiprocessor** (AMP) system. On an AMP system, one processor, the **master processor**, executes all operating system–related operations, such as making scheduling decisions and performing I/O. All other processors, called **slave processors**, perform operations allocated to them by the master processor. To take advantage of the Genesis MP platform, programs must explicitly call its programming interface to schedule threads for execution on its slave processors. For compatibility for applications designed to use the Genesis MP platform, Mac OS 8 provides asymmetric multiprocessor support.

The Mac OS 8 microkernel, however, provides full **symmetric multiprocessor (SMP)** support as well. On an SMP system like Mac OS 8, every processor on the computer executes its own copy of the operating system and communicates with the other processors as needed. Developers don't need to perform any special programming to take advantage of SMP systems. Instead, the operating system automatically schedules multiple tasks to execute simultaneously on all available processors. As a result, the multitasking capabilities of Mac OS 8 offer significant performance gains on multiprocessor computers.

A multithreaded application especially benefits from this performance gain. For example, a real-time task in a scientific application would get the use of one processor to handle data as it arrives from laboratory equipment, while another task might use a second processor to perform modeling operations based on that data. A third thread of execution might use yet another processor to make the results of the application's operations visible to researchers on networked computers.

Even on a multiprocessor computer, the cooperatively scheduled threads within a single task all execute on the same processor. That is, a task can't send its cooperatively scheduled threads to different processors the way that an application can send its separate tasks to different processors.

COMPATIBILITY NOTES

Genesis MP Platform Support

For the Genesis MP platform, DayStar Digital and Apple Computer collaborated to define a System 7.5 programming interface allowing developers to thread their programs. This programming interface is fully supported by Mac OS 8.

INTERPROCESS COMMUNICATION AND DATA SYNCHRONIZATION

In a multitasking environment, it's necessary for tasks to communicate. Tasks cannot move between processes. However, tasks using the interprocess communication mechanisms provided by Mac OS 8 can pass information to each other, even across different address spaces. A calendar application, for example, might consist of a cooperative program and a server program that share data. The cooperative program might allow the user to view and maintain a personal datebook. The server program might handle meeting proposals submitted by colleagues at network-connected computers. When it receives a meeting proposal, the server program passes this information to the cooperative program for display in the user's datebook.

Tasks sharing the same set of data must synchronize changes to the data. When the user of the networked calendar application schedules an appointment, for example, a task for the cooperative program must indicate that it's updating the user's calendar. If a colleague on the network simultaneously schedules an appointment with the user, a task for the server program must check for this indication. Finding that the calendar data's in use, the server program task must block its own execution. To resume execution, the blocked task needs to know when it's safe to update the user's calendar. The microkernel provides various services that allow tasks to synchronize their operations in this way.

Apple Events

Apple events are the most pervasive form of interprocess communication in Mac OS 8. An **Apple event** is a data structure used to direct the operation of, or communicate information to, a task. An Apple event contains a flexible hierarchy of additional data structures. This flexible hierarchy allows developers to share data between tasks at various levels of detail. For example, a meeting proposal can be sent across the network in an Apple event targeting a day, hour, and range of minutes within a user's calendar.

Apple events were introduced in System 7 so that applications could share services and information with each other. In System 7, only applications could use Apple events; other types of software, such as device drivers, could not. In Mac OS 8, all types of software can use Apple events. Communication can take place between tasks in the same process or in different processes, in the same address space or in different address spaces, on one computer or on connected computers. An OpenDoc part can also use Apple events to communicate with other parts and with tasks in any address space. An Apple event contains the identification of its destination, and the operating system delivers the Apple event to that destination.

In subsequent chapters, you'll become more familiar with the wide use of Apple events across the Mac OS 8 platform. Some of these uses include

- Event notification. Apple events are the chief means by which the operating system informs programs about user and system activity. For example, when the user chooses a command from the menu of an application, the operating system sends an Apple event to the application; this Apple event contains information needed by the application to respond to the user action.
- Scriptable automation. Programs that respond to Apple events can be controlled and automated via scripts created with scripting languages such as AppleScript. Using the Assistance Services, for example, a scriptable program can automate complex or seldom used operations for the user.
- Data sharing between programs and within a program. For example, a home finance application might use an Apple event to request a communications program to obtain current stock market information from an online service provider. The communications program, in turn, could return this information to the home finance application in an Apple event. Different tasks for the same program can share information in this manner.
- Synchronization between tasks. A task performing network I/O, for example, might send an Apple event to another task informing it that a file has been successfully sent across a network.

A **script** is a series of statements, written in a scripting language, instructing a computer to perform various operations. **Scripting languages** are designed to automate and control programs and to be easier to learn and use than complex languages like C.

COMPATIBILITY NOTES

Apple Events, High-Level Events, and PPC Toolbox Services

Mac OS 8 fully supports the programming interfaces defined by the System 7 Apple Event Manager.

Mac OS 8 supports System 7 high-level events, but only the main tasks of cooperative programs can send high-level events other than Apple events. Main tasks for cooperative programs can also use the PPCBrowser mechanism and the PPC Toolbox functions, but all other tasks can use only the PPC Toolbox functions.

Apple events are faster and more flexible than other high-level events or the PPC toolbox services, which Mac OS 8 supplies only for System 7 application compatibility.

Low-Level Interprocess Communications

In addition to Apple events, developers can use shared data, shared memory areas, and the Microkernel Messaging Service for interprocess communication. Apple events are sort of the *lingua franca* of Mac OS 8, permitting the operating system, programs, and scripts to communicate with each other locally and across networks according to a well-established messaging protocol. By comparison, shared data, shared memory areas, and the Microkernel Messaging Service require developers to establish and follow their own conventions for using these low-level forms of interprocess communication.

Shared Data

Shared data is available to multiple tasks in the same process. As explained in the previous chapter, different tasks for the same program share the same memory areas for dynamic storage allocation. Tasks can store shared data in these memory areas. Two tasks in the same process, for example, can share a single set of global variables for communication and synchronization purposes.

Shared Memory Areas

A task can create a **shared memory** area to share its data with a task in another address space. If programs in different address spaces share a large amount of data, especially when that data is continually updated, a shared memory area is likely to be a more efficient mechanism than Apple events for distributing that data among tasks. A shared memory area can begin at the same address in various address spaces (which is useful if the tasks sharing the data refer to it by pointers), or it can reside at different addresses. A shared memory area can have different access permissions in different address spaces; for example, a program might write data into a shared memory area in its own address space but make the data read-only to programs in other address spaces, preventing other programs from corrupting the data.

Microkernel Messaging Service

If for some reason a developer cannot or prefers not to use Apple events, the **Microkernel Messaging Service** is available for transporting data from one task to another, typically between different processes. The messaging service allows bidirectional data transfer so that data may be part of a message, and additional data may be returned in the reply. It's up to developers to establish their own conventions for interpreting the information exchanged with this service.

Data Synchronization Among Tasks

Two or more tasks sharing information must synchronize their access to that data. Otherwise, two tasks independently making changes to the same data might corrupt its integrity. Mac OS 8 offers synchronization mechanisms to protect the data shared among tasks.

Remember that cooperatively scheduled threads don't need to use these synchronization mechanisms, because one cooperatively scheduled thread can't be preempted by another thread within the same task. A developer simply needs to ensure that each cooperatively scheduled thread yields control only *after* making changes to any data shared by other threads.

For these synchronization mechanisms to protect program data, a program must observe synchronization conventions. For example, if one task holds a lock to particular data, another task must not modify the data until it has acquired the lock. Locks and other synchronization mechanisms are described in the next sections.

Locks

A **lock** is a data structure used to synchronize access to a shared resource such as the contents of memory locations. Only the task holding a lock is allowed to modify the data associated with the lock. A **simple lock** prevents other tasks from acquiring the lock until the task holding it has released it. A **read/write lock** allows one or more tasks to acquire the lock for the purpose of simultaneously reading data, but this type of lock allows no more than one task to modify the data at a time.

Counting Semaphores

A **counting semaphore** is a synchronization mechanism containing a count variable that may be equal to or greater than zero. Multiple tasks can increment and decrement a counting semaphore. Typically, a task will test whether the counting semaphore is greater than zero before performing some action involving a shared resource.

Microkernel Queues

A **microkernel queue** can be used as a synchronization mechanism or a very simple interprocess communication mechanism. One or more tasks use a microkernel queue to notify another task of some occurrence, for instance, the completion of an asynchronous operation. The task being notified examines the microkernel queue for the notification; this task may, for example, block until the notification appears. A microkernel queue can hold multiple notifications.

Communication with this mechanism takes place in one direction only; that is, the tasks writing to a microkernel queue don't receive replies from the task reading the queue.

System Notification Service

The **System Notification Service** allows one task to broadcast information about a change in the state of the system. Any number of other tasks can subscribe to its notifications. For example, the device driver for a display screen can use the System Notification Service to announce that the user has changed screen resolutions or bit depth. Programs relying on the resolution or color capabilities of the device can then take action based on the notification.

Atomic Operations

An **atomic operation** is a simple routine—such as one that increments or decrements a value, tests and sets a value, or compares and swaps values—that executes to completion; it cannot be interrupted. In Mac OS 8, these operations are implemented using instructions provided by the CPU.

Event Groups

An **event group** consists of bits that can be set individually or in different combinations and used to signal client tasks. Developers can use atomic operations and event groups to implement their own sychronization mechanisms.

COMPATIBILITY NOTES

The Disabling of Interrupt Handlers in System 7

Most applications don't deal with synchronization in System 7, where the current application can always assume that it controls the computer. When they need to synchronize with code running at interrupt level, some applications compiled for 68K-based Macintosh computers disable the execution of completion routines (such as for file I/O operations and vertical retrace interrupts) by disabling the execution of all hardware interrupt handlers across the system. When 68K applications attempt to synchronize data access in this way, Mac OS 8 disables the execution of completion routines while allowing all hardware interrupt handlers to continue to execute. In Mac OS 8, only device drivers are able to disable hardware interrupts. Although this method of synchronization is available to device driver writers, Apple Computer generally recommends against using it.

SUMMARY

In the same way that multitasking makes the system more efficient, multithreading makes an application more efficient, which helps the user be more productive. For example, by performing user interface operations in one thread of execution and time-consuming network I/O operations in another thread of execution, a multithreaded application enables the user to continue working with the application without waiting for lengthy network operations to complete.

Multithreaded applications use interprocess communication to share information. For example, one task for a statistical simulation program could perform a time-consuming calculation in the background. When finished with this calculation, the task could send its result in an Apple event to the main task of the program.

The multithreading capabilities of Mac OS 8 offer significant performance gains on multiprocessor computers. For example, while an application uses one processor to perform animation rendering calculations, it might use another processor to perform real-time video capture, and use yet a third processor to simultaneously serve the results of its operations to users on the World Wide Web.

PLANNING A PRODUCT FOR MAC OS 8

If you're a developer, you can take the following steps now to prepare a product that takes advantage of multithreading:

1. Consider whether your program consumes very much processing time when it's not interacting with the user. If it does, separate your code into components that perform user interface tasks, computations, and I/O operations. You can then implement these components more easily as separate threads of execution in a multithreaded program.
2. Make your existing application AppleScript scriptable. This will prepare your application to use Apple events for interprocess communication.

6 The Virtual Memory System

Mac OS 8 supplies the addressable memory necessary for multiple large programs to be open simultaneously, and its multitasking architecture allows these programs to share the CPU efficiently. So that the user may open and run more programs than can fit in physical memory, the operating system provides a highly efficient form of virtual memory.

The virtual memory system extends the amount of physical memory by using secondary storage, such as a hard disk, to store code and data not immediately needed by the CPU. The virtual memory system thereby keeps portions of physical memory free at all times. When the CPU requires code or data that's not in physical memory, the virtual memory system transfers the needed portions from secondary storage into physical memory.

The Mac OS 8 virtual memory system makes efficient use of secondary storage by using only enough disk space to support currently open programs. For example, the virtual memory system dynamically allocates only enough disk space to hold unneeded portions of temporary program data, and it allocates no disk space at all for program code. For holding unneeded portions of program code, the virtual memory system simply uses the executable files already stored on disk.

Users and the developers of application-level software needn't do anything special to take advantage of this virtual memory system. Always in operation, this system is an integrated, core feature of Mac OS 8.

Key Terms and Concepts

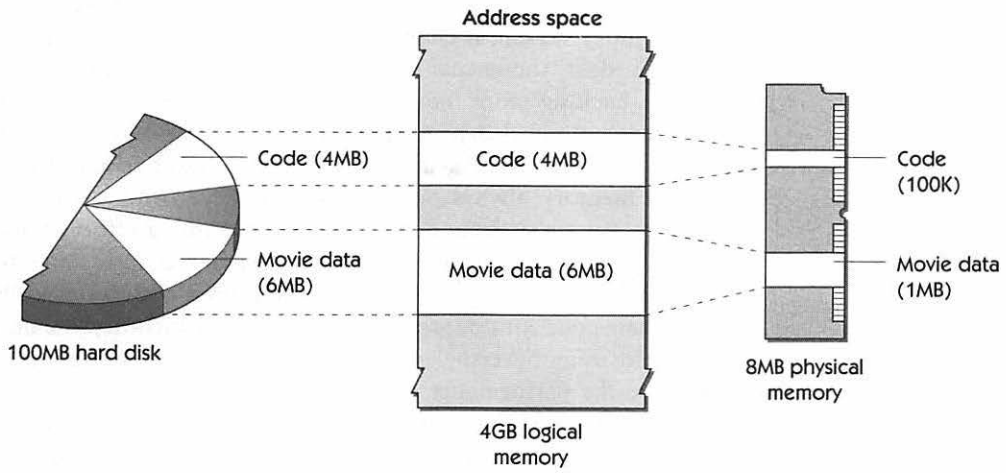
- **Virtual memory** is addressable memory beyond the limits of available physical memory. Mac OS 8 extends physical memory by storing on a secondary storage device, such as a hard disk, code and data not immediately required by the CPU.
- A **page** is the smallest unit, measured in bytes, of information that the virtual memory system can transfer between physical memory and backing store. **Paging** is the transfer of pages between physical memory and backing store.
- **Backing store** is a repository—typically a file on a secondary device such as a hard disk—for pages of code or data that aren't currently in physical memory.
- The **backing provider** is the operating system service responsible for managing pages of physical memory and transferring code and data between backing store and physical memory in response to page faults.
- A **page fault** is an exception that causes a page of data or code needed by the CPU to be read from backing store into physical memory.
- A **scratch file** is a temporary file used as backing store for a memory area, such as for a stack or dynamic storage allocation, containing data not associated with a permanent disk file. A scratch file grows and shrinks dynamically in response to system demands.
- A **memory-mapped file** is a disk file whose contents are mapped into a memory area. The backing provider transfers portions of these contents from the file's permanent location on disk to physical memory as needed in response to page faults. Thus, the disk file (instead of a separate scratch file) serves as backing store for the code or data not immediately needed in physical memory.

Major Points of Interest

The virtual memory system allows the user to open and run more programs than can fit in physical memory. Although system performance is optimal when there is enough physical memory to hold all of the code and data needed by all open programs, it's impractical for most users to have enough physical memory to satisfy all possible demands.

Figure 6.1 illustrates how the virtual memory system extends physical memory while making efficient use of the hard disk space used for backing store. Suppose that the user has installed a digital-video editing application containing 4MB of code onto the hard disk. When the user launches that application, the operating system creates a process for it within a 4-GB address space. The operating system maps all of the code for that application

FIGURE 6.1 The association of disk files, memory areas, and physical memory



from its disk file into a system-wide memory area. However, the CPU doesn't need access to all of that code at once. So the virtual memory system transfers into physical memory only the code immediately needed by the CPU. In this figure, that amounts to 100K of code. When any portion of this code is no longer needed, the virtual memory system releases it from physical memory.

Suppose then that the user begins editing video data with this application. The application can use the virtual memory system to memory-map this data. If the data amounts to 6MB on disk, then the virtual memory system creates a 6-MB memory area in the application's address space. The CPU, meanwhile, doesn't need all of that data to be available in physical memory. So again, the virtual memory system transfers into physical memory only the data immediately needed by the CPU. In this figure, that amounts to 1MB of data.

When launching a program, the operating system automatically creates a memory area for the program's tasks to store temporary data, such as the global variables and dynamic data structures used by the program at execution time. The virtual memory system dynamically allocates space from a scratch file on disk to serve as backing store for this memory area. As these backing store needs grow, the virtual memory system allocates more disk space for the scratch file; as these backing store needs diminish, the virtual memory system releases disk space.

The number and size of open programs are constrained only by the amount of disk space available for scratch files. The virtual memory system never allocates disk space to serve as backing store for code. Instead, the virtual memory system always memory-maps the disk files of code. Thus, the disk file (instead of a separate scratch file) serves as backing store for the code not immediately needed in physical memory. Programs can also use memory-mapped files to gain access to data stored on disk.

Users and developers needn't do anything special to take advantage of this virtual memory system. If code or data isn't already in physical memory when the CPU needs it, the virtual memory system automatically pages the code or data from backing store into physical memory. The virtual memory system either releases unneeded pages from physical memory or, when necessary to save data that's been changed, writes these pages to backing store.

Virtual memory always operates in Mac OS 8, and it works for application-level software without any special programming effort by its developers. (Some device drivers and other types of privileged code can't tolerate page faults, so Mac OS 8 supplies a special programming interface allowing them to keep their code and data resident in physical memory.) A developer with special needs may nevertheless give "hints" in a program about memory use to optimize the performance of the virtual memory system. For example, an application needing a large amount of multimedia data for a future operation can ask the virtual memory system to asynchronously transfer pages of memory from backing store into physical memory. The application might also tell the system when data won't be referenced again, allowing Mac OS 8 to reuse those pages immediately.

The major components of the Mac OS 8 virtual memory system are the microkernel and the backing provider. The microkernel creates memory areas for processes and recognizes page faults within those areas. The backing provider associates backing store with memory areas, and the backing provider resolves the page faults by transferring code and data between backing store and physical memory.

MAC OS HERITAGE

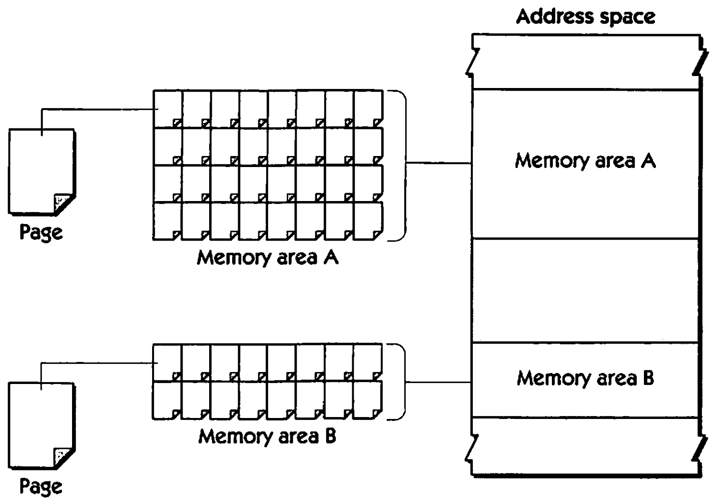
Virtual Memory in System 7

In System 7, the user selects whether to use virtual memory, and how much to use, through the Memory control panel. These selections don't take effect until the user restarts the computer, at which time System 7 allocates the user-specified amount of virtual memory and its corresponding disk space. When the user launches applications, the Process Manager in System 7 allocates fixed-size memory partitions from this single, logical address space.

In Mac OS 8, by contrast, virtual memory is always on—the user doesn't select whether to use it—and it's dynamically allocated—the user doesn't need to specify a fixed amount for it.

System 7 doesn't allow its system heap to be paged to disk; instead, the system heap remains in physical memory. (The **system heap** is an area of memory reserved for data structures used by the operating system in support of System 7 applications.) In Mac OS 8, however, almost all of the system—except for critical portions like the microkernel and the file system—is pageable. This makes more physical memory available to application-level software.

FIGURE 6.2 The organization of memory in an address space



THE ORGANIZATION OF VIRTUAL MEMORY

The virtual memory system transfers code and data between the CPU and backing store in page-sized portions of information. A memory area consists of a set of pages. A set of memory areas, in turn, constitutes an address space, as illustrated in Figure 6.2. (The size of a page, measured in bytes, is set by the CPU architecture. For the current generation of PowerPC processors, that size is 4K.)

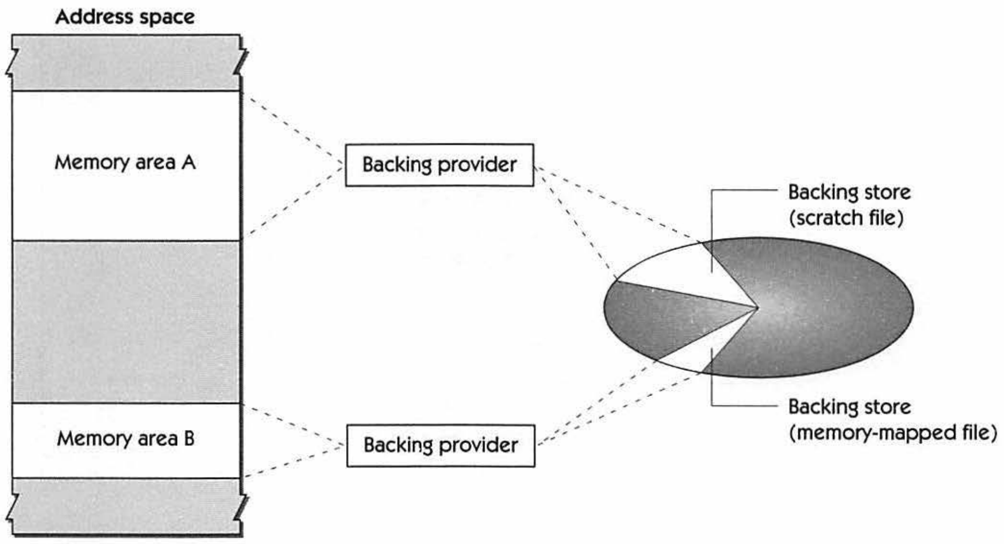
From the perspective of the virtual memory system, there are three types of memory areas:

- resident memory areas
- memory areas associated with scratch files
- memory areas associated with memory-mapped files

The pages of resident memory areas are always kept in physical memory. The microkernel and other critical portions of the operating system are kept in resident memory areas.

Those memory areas associated with scratch files are generally used by programs for their dynamic storage-allocation needs. To make efficient use of the disk space needed as backing store for these types of memory areas, the virtual memory system dynamically allocates only enough disk space to satisfy the

FIGURE 6.3 The mapping between memory areas and backing store



immediate needs of that program. For example, imagine that Memory area A in Figure 6.3 is used by a program for the storage of global variables, data structures, and other data that can't be determined until the program is running. The backing provider dynamically allocates space from a scratch file just large enough to serve as backing store for this memory area.

Memory areas for memory-mapped files have no scratch files associated with them. Suppose then that the program in Figure 6.3 opens a file and requests that the virtual memory system open it as a memory-mapped file. In the act of memory-mapping this file, the virtual memory system creates Memory area B, and the backing provider maps the contents of the disk file into Memory area B.

When the CPU tries to access a page of data or code not currently located in physical memory, the CPU generates a page fault. In response to this page fault, the backing provider reads the necessary page or range of pages from backing store into physical memory.

To summarize, then, the operating system when instantiating a process creates memory areas that constitute only a small portion of an address space. The virtual memory system allocates disk space only for the storage of temporary data not immediately needed by the CPU. At any given time, the virtual memory system might load only a small amount of code and data from backing store into physical memory. In this way, even with 4GB of addressable memory, a process can require very little additional backing store and only a small amount of total physical memory.

The rest of this chapter describes these elements of the virtual memory system in greater detail.

MAC OS HERITAGE

The Preallocated Address Space in System 7

In System 7, the entire size of its single address space is set at boot time when the user selects the amount of virtual memory with the Memory control panel. Mac OS 8 allocates and releases memory from multiple address spaces dynamically in response to the system's memory needs.

THE BACKING PROVIDER

The microkernel is responsible for recognizing page faults, which occur when information currently in backing store needs to be read into physical memory. It's up to the backing provider to resolve page faults. Whenever a program or the operating system creates a pageable memory area (that is, a memory area that needn't remain resident in physical memory), that area is associated with the backing provider. When a page fault occurs in any memory area, the microkernel alerts the backing provider. The backing provider then reads the necessary pages of data or code from backing store into physical memory.

The backing provider dynamically allocates scratch files for memory areas containing temporary data. For code stored in files on disk, the backing provider uses the disk files themselves as backing store. The backing provider can also use memory-mapped data files as backing store at the request of programs.

The multitasking capabilities of Mac OS 8 enhance the efficiency of the virtual memory system. When a task is blocked on a page fault (for instance, when a task waits for data to be paged from a disk into physical memory), the microkernel switches to another task, allowing the system to perform other operations until the backing provider finishes processing the page fault.

MAC OS HERITAGE

System 7 Swap Space

Swap space, sometimes called **scratch space**, is a single, preallocated area of a hard disk used by some operating systems, such as System 7 and some variants of UNIX, for the temporary storage of code and data that have been paged out of physical memory. With System 7 virtual memory, data files are, by default, paged to and from the swap space. For example, when a data file is opened by an application, the file is typically read from disk into physical memory. If the file is too large to fit into physical memory, unused portions of the file are paged to swap space on the disk. Thus two copies of the file exist: one copy, the original, is stored on disk in the file system; the other copy is held partially in physical memory and partially on disk in swap space.

By comparison, a memory-mapped file in Mac OS 8 exists in only one location on disk. Portions of that file are copied into physical memory when needed by the CPU. Unneeded portions can either be released from physical memory or saved back to the original disk file, but they're never paged to a separate swap space.

MEMORY-MAPPED FILES

When the operating system or a program opens a file, the contents of that file can be mapped into a new memory area so that the disk file serves as backing store for its own representation in physical memory. This association of a disk file to a memory area is called **memory mapping**. Memory mapping conserves the amount of disk space needed for backing store. Memory mapping also allows multiple tasks to share one copy of a disk file in memory, and in this way, memory mapping conserves physical memory. For example, the code for all operating system services is memory mapped into system-wide memory areas so that a single copy of each of these services can be used by all programs running on the system.

At the request of a program opening a data file located on disk, the virtual memory system can open a memory-mapped version of that file. After the operating system creates a memory area for that file, a backing provider maps the contents of the file into this memory area. After opening a memory-mapped data file, such as a user document, a program can easily read the document by accessing the logical addresses within its memory area instead of, say, streaming the file from disk through buffers.

When multiple programs open the same data file, whether by memory mapping it or streaming it, the operating system gives them all access to the same data. For developers, this simplifies the task of designing programs that share data.

The memory areas into which Mac OS 8 maps code are given read-only access privilege. Because code can't be changed in memory, the virtual memory system never writes it back to disk, allowing the I/O system to handle other demands. When a piece of code is no longer needed, the virtual memory system releases it from physical memory. If that code is needed again, the virtual memory system simply reads it back from the disk file.

In the same way, memory-mapped data files in read-only memory areas are never written back to disk. Pages of a memory-mapped data file are written back to disk only when both of these conditions are met:

- the pages have been changed
- either the virtual memory system wants to make more physical memory available, or a program asks the virtual memory system to save those pages to disk

MAC OS HERITAGE

Memory-Mapped Files for System 7 on PowerPC-Based Computers

System 7.1.2 began using memory-mapped files for PowerPC code. As in Mac OS 8, code in these memory-mapped files are read-only. All data, however, is paged to and from swap space in System 7.1.2 and subsequent versions of System 7.

SCRATCH FILES

When a program is launched, Mac OS 8 acquires various operating system resources necessary for the program and bundles them into a process. These resources include memory areas for the program's temporary data. The backing provider maps these memory areas to space dynamically allocated from a scratch file. When the operating system releases these memory areas, such as when a process terminates, the backing provider relinquishes the space allocated from the scratch file. In this way, disk space for the scratch file is dynamically reclaimed whenever it's no longer needed for backing store.

If more than one disk is mounted to a system, the user can specify which disk should contain the scratch file. For example, if a disk contains video data being edited by the user, the user might select another disk to hold the scratch file in order to improve the I/O performance of the disk containing video data. The user can also divide temporary backing store into several scratch files located on different disks.

SUMMARY

Unlike System 7, where virtual memory is an option that the user chooses to use and configure, Mac OS 8 uses a virtual memory system that's dynamic, automatic, and always in operation. The virtual memory system always keeps portions of physical memory free by releasing or writing to disk pages not needed by the CPU. When the CPU does need code or data not currently in physical memory, the virtual memory system pages it from backing store into physical memory. Shuffling pages between physical memory and backing store as needed by the CPU, the virtual memory system provides execution support for the many programs that can exist at any time on a Mac OS 8 system.

To make the most efficient use of the storage capabilities of a user's system, the virtual memory system creates scratch files for backing store only when needed, and the virtual memory system releases this backing store when it's no longer needed. The virtual memory system doesn't create any space as backing store for memory-mapped files. Instead, the virtual memory system uses the disk files themselves as their own backing store. All files containing code, for example, are memory mapped.

COMPATIBILITY NOTES

System 7 Virtual Memory Manager

System 7 developers should note that although virtual memory is always in effect in Mac OS 8, any call to the Gestalt function using the System 7 selector gestaltVMAttr indicates that virtual memory is off. This is to preclude System 7 applications from performing the special actions necessary to take advantage of System 7 virtual memory. These actions are unnecessary—even undesirable—in Mac OS 8. Nevertheless, Mac OS 8 supports software that uses any of the functions defined by the System 7 Virtual Memory Manager—with the lone exception of the LockMemoryContiguous function, which isn't supported by Mac OS 8.

PLANNING A PRODUCT FOR MAC OS 8

If you are a developer, you can take the following steps to prepare products that take advantage of the Mac OS 8 virtual memory system:

1. If you've already developed a System 7 application, ensure that it operates well in the System 7 virtual memory environment.
2. Consider whether and how your application can take advantage of memory-mapped data files.

7 Dynamic Storage Allocation

Nearly all code uses memory areas to store data, such as global variables and various data structures, that can't be determined until the code is running. Mac OS 8 supplies a very fast Dynamic Storage-Allocation Service so that code can create, allocate storage from, expand, and delete memory areas for its temporary data. The memory-allocation services of System 7 and earlier versions of the Mac OS make efficient use of physical memory, but this memory efficiency comes at the expense of the speed of application performance. The virtual memory system of Mac OS 8, as you saw in the previous chapter, makes very efficient use of physical memory, thereby allowing the Mac OS 8 Dynamic Storage-Allocation Service to be optimized for speed instead of memory efficiency.

The Dynamic Storage-Allocation Service in Mac OS 8 is reentrant and can be used by any type of code. Mac OS 8 also supports the System 7 Memory Manager, a cooperative service available for application-backward compatibility. By following guidelines relating to the use of a subset of Memory Manager routines, developers can, with minimal revision, make their existing System 7 products perform faster by using the Dynamic Storage-Allocation Service supplied by Mac OS 8.

Key Terms and Concepts

- The **Dynamic Storage-Allocation Service** defines a programming interface by which code—such as an application or a device driver—manages memory allocations for its data storage needs. The Dynamic Storage-Allocation Service supplies memory allocators that implement this programming interface, but developers can choose to create memory allocators of their own.
- A **memory allocation** is a range of logical addresses used for storing a particular piece of data, such as a global variable or a data structure. A memory allocation can range in size from 1 byte to multiple pages.
- A **memory allocator** is a shared library used by client code for creating, expanding, and deleting memory areas and for acquiring memory allocations from these areas. (A **shared library** is a set of routines or static data that can be called by multiple programs. Shared libraries are discussed in the next chapter.) As memory areas become fully allocated, a memory allocator automatically creates new memory areas and thereby supplies additional storage to its client code. Mac OS 8 provides three memory allocators: a per-process memory allocator, a system-wide memory allocator, and a nonpageable-memory allocator. For any special needs, Mac OS 8 developers can create their own memory allocators.
- A **per-process memory allocator** is instantiated by Mac OS 8 for every process, and every process uses its instantiation of this allocator to supply most of its memory-allocation needs. For example, when a Mac OS 8 program requests the operating system to provide storage for data structures related to windows, a per-process memory allocator supplies the necessary memory allocation. Typically, all tasks in a process use the per-process memory allocator instantiated for that process.
- The **system-wide memory allocator** manages memory allocations in system-wide memory areas. Any type of code, nonprivileged or privileged, can use this allocator.
- The **nonpageable-memory allocator** is instantiated once upon system startup. Only privileged code can use this memory allocator, which keeps all of its memory allocations resident in physical memory for the benefit of privileged code—such as a hardware interrupt handler for a device driver—that can't tolerate page faults.
- The **Memory Manager** is a cooperative service used by System 7 applications to dynamically acquire and release memory allocations.
- The **forward-compatible memory guidelines** specify the use of a subset of Memory Manager functions; applications adhering to these guidelines cause the operating system to invoke the per-process memory allocator instead of the System 7 Memory Manager. By following these guidelines, the developer of a System 7 application can, with minimal

revision of the application, attain much better performance from the application.

- A **stack** is a memory area where a task stores some of its temporary variables during execution. Every task has its own stack. For example, when a task calls routines, their parameters, local variables, and return addresses may be loaded onto a stack. Compilers typically generate the code that manages the stack for a task. As with a stack, register manipulation is typically performed automatically for compiled code. However, for temporary data such as global variables and data structures not stored on a stack or in a processor register, storage isn't automatic. Instead, a task must acquire storage for such data. The storage of such data is the topic of this chapter.

MAJOR POINTS OF INTEREST

During execution time, nearly every type of code needs to store data temporarily in memory locations within its address space. Chapter 3 described how each address space in Mac OS 8 consists of 4GB of logical addresses, and Chapter 6 described how the virtual memory system uses backing store to support such a large amount of addressable memory. Within every address space, up to 3GB of address locations are either used to store code for programs, device drivers, and the operating system or are reserved for other uses by the operating system. Of all logical addresses within an address space, Mac OS 8 makes at least 1GB available to programs for temporary data storage.

In Mac OS 8, applications can use either the Dynamic Storage-Allocation Service or the Memory Manager to supply data storage. The Dynamic Storage-Allocation Service supports preemptive scheduling and can be used by any type of code; the Memory Manager is a cooperative service that supports System 7 application compatibility.

The Dynamic Storage-Allocation Service defines a programming interface by which code, such as an application or a device driver, can acquire, expand, and release memory allocations for data storage. This programming interface allows programmers to use pointers when referring to memory allocations. By comparison, System 7 applications using the Memory Manager programming interface refer to memory allocations through handles. Developers who have used the Memory Manager's handle-based routines—especially those designed to reduce heap fragmentation—will find that the Mac OS 8 pointer-based programming interface offers an easier programming model as well as significantly improved performance.

A **pointer** is a variable containing the address of a byte in memory. A **handle** is a variable containing the address of a nonrelocatable pointer, which in turn refers to the address of a relocatable block of data.

Memory allocators implement the code that actually reserves and releases memory locations on behalf of clients of the Dynamic Storage-Allocation Service. Memory allocators dynamically increase the storage available to their clients as allocation needs increase. Mac OS 8 supplies memory allocators that fit the needs of most developers.

A memory allocator is implemented as a type of shared library known as a **plug-in.** Plug-ins, as described in the next chapter, can be prepared and released as needed through the programming interface defined by the Code Fragment Manager. This plug-in architecture adds flexibility to the Dynamic Storage-Allocation Service—developers with special needs can supply their own memory allocators and easily integrate them with the operating system.

All Mac OS 8 memory allocators are reentrant, so they can be used by any type of code—including server programs, device drivers, and cooperative programs. By using the memory allocators, all code gains a significant performance boost compared to code using the handle-based Memory Manager. With little revision of existing System 7 applications, however, developers following the forward-compatible memory guidelines can easily take advantage of the Mac OS 8 memory allocators even when using the routines of the Memory Manager.

MAC OS HERITAGE

The Memory Manager

The Memory Manager was initially designed to allow application software to run on computers with only 128K of physical memory. In this environment, efficient use of physical memory was essential. So the Memory Manager was designed to constantly load blocks of data into and out of physical memory, dynamically grouping these blocks together so that the application heap wouldn't be broken into fragments of addressable memory too small to be of use.

Blocks of data moved by the Memory Manager are referenced by handles. Whereas the Memory Manager may move a block, it won't move the location of the pointer to that block, always allowing both the Memory Manager and an application to refer to that block through its handle.

The virtual memory system of Mac OS 8 makes very efficient use of physical memory, so the overhead associated with the creation and maintenance of handles and the shuffling of data within physical memory are no longer necessary. As a result, the Mac OS 8 pointer-based memory allocators perform much faster than the handle-based System 7 Memory Manager.

DYNAMIC STORAGE-ALLOCATION SERVICE

The Dynamic Storage-Allocation Service defines a pointer-based programming interface. Any type of code can use this programming interface to request memory allocations for storing the code's temporary data. To assist developers in migrating their System 7 code while improving its performance under Mac OS 8, this programming interface allow developers to use handles as well as pointers when referencing memory allocations.

The Dynamic Storage-Allocation Service also supplies a set of memory allocators. On behalf of client code, a memory allocator creates memory areas for data storage, and it acquires memory allocations from these areas in response to requests from the client code. As soon as all the storage in one memory area is allocated, the memory allocator creates another memory area. The memory areas created by an allocator are usually discontiguous—that is, they are usually separated throughout an address space.

Typically, a program uses the per-process memory allocator instantiated for it by the operating system. However, it can also use the system-wide memory allocator, and it can also use its own memory allocators. A program can thereby request the services of multiple memory allocators, each optimized for a special use.

A piece of privileged code, such as a device driver, typically uses the microkernel's instantiation of the per-process memory allocator. Nonprivileged code is denied any access to the data storage created by this instantiation of the per-process memory allocator. Whenever it needs to grant nonprivileged code access to its data, privileged code can use the system-wide memory allocator. Whenever privileged code can't tolerate page faults, it can use the nonpageable-memory allocator. Privileged code can also use any custom memory allocators that it might supply.

These various memory allocators are described next.

Recall that only the microkernel portions of device drivers, and certain other portions of the operating system contain **privileged code.** To protect the stability of the base operation system, most code is **nonprivileged**, meaning that it's restricted from critical CPU instructions, hardware addresses, and much of the data used by privileged code.

Per-Process Memory Allocators

When instantiating a process, the operating system also instantiates a per-process memory allocator for that process. Tasks within a process generally use this per-process memory allocator. For example, when a Mac OS 8 program asks the operating system to allocate storage for data structures related to files or windows, the per-process memory allocator for that program typically supplies the necessary memory allocations.

When the operating system instantiates a per-process memory allocator for nonprivileged code, the allocator creates a memory area from which it acquires storage at the request of that code's tasks. As you will see in the next chapter, the Code Fragment Manager places the global variables and other static data for a process into this memory area. This memory area resides exclusively in the address space of the process, and all of the process's tasks have read/write access permission to the contents of this area. The virtual memory system creates a scratch file for this memory area so that the virtual memory system can page its contents. If the memory area becomes fully allocated, the memory allocator creates another memory area in the same address space. To fulfill memory-storage needs, the memory allocator continues creating additional memory areas—at least until the operating system determines that a programming error has led to excessive and potentially catastrophic requests for memory allocations.

Any privileged code associated with the microkernel typically uses the per-process memory allocator instantiated for the microkernel. All nonprivileged code is limited to read-only access to the memory areas created by this instantiation of the allocator, and all of its memory areas are pageable. Because the microkernel operates in all address spaces, the memory areas used by this instantiation of the per-process memory allocator are also system-wide.

A program can also create and use additional instances of the per-process memory allocator. For example, it might be useful for a cooperative program to use a second instantiation of this allocator for maintaining memory areas dedicated to a data cache—when the cached data becomes stale, for instance, the program can quickly and easily release its storage from all memory areas in a single operation.

The per-process memory allocator is designed to take optimum advantage of the virtual memory system and to acquire storage in sizes optimal for most types of code. For nearly all developers' needs, the per-process memory allocator is well suited. However, developers can use additional memory allocators if necessary.

The System-Wide Memory Allocator

Any type of code, nonprivileged and privileged, can use the system-wide memory allocator to acquire, expand, and release data storage from pageable, system-wide memory areas. The memory areas created and used by this allocator appear at the same locations in every address space. These areas also offer read/write permission to all code, making this allocator useful when code stores data that needs to be accessible at identical locations to all tasks in every address space. These areas actually have limited use for data sharing, however, because any task can potentially corrupt any portion of the contents of these areas. Instead, as explained in Chapter 5, a shared memory area is generally more useful for data sharing.

The Nonpageable-Memory Allocator

The Dynamic Storage-Allocation Service provides a third memory allocator. Like the system-wide memory allocator, the nonpageable-memory allocator manages memory allocations in system-wide memory areas. However, this memory allocator is available only to privileged code (its memory areas are read-only to nonprivileged code), and the microkernel holds the contents of its memory areas in physical memory at all times. When privileged code—such as a hardware interrupt handler for a device driver—cannot tolerate a page fault, it uses this memory allocator.

COMPATIBILITY NOTES

The Pool Manager

The Pool Manager defines a programming interface used by privileged code in System 7 for making memory allocations. For example, developers of device drivers on occasion need to use the Pool Manager in System 7. The Mac OS 8 memory allocators make the Pool Manager unnecessary, and Apple Computer encourages developers of privileged code to adopt the Mac OS 8 Dynamic Storage-Allocation Service in lieu of the Pool Manager. However, for backward compatibility, the Dynamic Storage-Allocation Service supports the programming interface defined by the Pool Manager.

Custom Memory Allocators

The memory allocators supplied by Mac OS 8—the per-process memory allocator, the system-wide memory allocator, and the nonpageable-memory allocator—meet most developer needs. However, developers with special needs can supply their own custom memory allocators. For example, it might be useful for a hardware vendor to write a device driver that allocates and releases storage from physical memory on a card instead of from system-wide memory areas maintained by the operating system.

A developer can ascribe various attributes to a custom memory allocator, such as whether

- it's optimized to acquire memory allocations of a certain size—such as 1 or 2 bytes or multiple numbers of pages
- its memory areas are private to an address space or system-wide and available to all address spaces
- its memory areas are pageable or must remain resident in physical memory
- the microkernel should allocate backing store space for pageable memory areas only as needed, or preallocate backing store space for entire memory areas as they're created
- its memory areas should have separate access permissions for privileged and nonprivileged code

MAC OS HERITAGE

Dynamic Memory Allocation in System 7

In System 7, operating system software, device drivers, desk accessories, system extensions, and applications can dynamically allocate only from memory areas known as the system heap and application heaps. System 7 also offers a temporary memory scheme, but because the amount of that memory continually changes during execution time, programs can never rely on a fixed amount, making temporary memory mainly useful for preventing programs from crashing by supplying emergency, short-term memory allocations.

THE MEMORY MANAGER FOR SYSTEM 7 APPLICATIONS

For compatibility with older applications, the operating system also supports the System 7 Memory Manager—a cooperative service for dynamically acquiring and releasing memory allocations from memory areas known as application heaps. An **application heap** is a memory area assigned exclusively to a System 7 application for the application's temporary data-storage needs. Whereas the Mac OS 8 memory allocators dynamically create additional memory areas to fulfill a program's storage needs, an application heap is fixed in size at application launch time and can't be expanded.

When a System 7 application is launched in Mac OS 8, the operating system creates an application heap for the program. The operating system also instantiates a per-process memory allocator, which allocates storage for the program's static data, such as the global variables used by the program code. In addition to memory areas created for the application heap and the per-process memory allocator, the operating system also maintains a memory area within the cooperative program address space called the system heap. The **system heap** is reserved for storing various data structures used by the Process Manager and other portions of the operating system in support of System 7 applications. When System 7 applications in Mac OS 8 allocate memory for windows, for example, the operating system stores various data structures in the system heap.

Since the System 7 Memory Manager is a cooperative service, its programming interface can be called only by the main tasks of cooperative programs. Because of the way that the Memory Manager moves and tracks memory allocations, developers must implement memory-conservation measures in their own code. In addition to providing faster performance, use of the Mac OS 8 memory allocators greatly reduces memory-management overhead for developers.

To improve the performance of programs built on a System 7 code base, Apple Computer has designed an interface between the Memory Manager and the Dynamic Storage-Allocation Service. By following the forward-compatible memory guidelines, a developer can easily adapt a System 7 application so that the application uses a per-process memory allocator for its dynamic storage needs.

COMPATIBILITY NOTES

System 7 Memory Manager

Over the years, many Macintosh developers have taken idiosyncratic approaches to using the Memory Manager. Because of the number of changes in the addressing model introduced by the Mac OS 8 virtual memory system, developers who haven't strictly followed Apple Computer guidelines to the use of the System 7 Memory Manager may need to revise their System 7 applications to run compatibly in Mac OS 8.

Forward-Compatible Memory Guidelines

The forward-compatible memory guidelines provide a link from the System 7 Memory Manager programming interface to the per-process memory allocator available with Mac OS 8. These guidelines specify a subset of the routines defined by the System 7 Memory Manager. If a System 7 application uses only this subset of Memory Manager routines and programmatically makes this known to the operating system, the operating system automatically invokes the per-process memory allocator instead of the Memory Manager. This allows System 7 applications to continue using System 7 calls that take memory handles as parameters and yet, with very little revision, perform significantly faster than they do under System 7.

In Mac OS 8, dynamic storage needs for OpenDoc parts are handled by the per-process memory allocator. System 7 OpenDoc part editors using the OpenDoc memory management protocol instead of the System 7 Memory Manager are already forward compatible with the per-process memory allocator available with Mac OS 8.

COMPATIBILITY NOTES

A5 Worlds

For A-trap–based System 7 applications compiled to run on 68K CPUs, Mac OS 8 emulates the **A5 world**—the area of memory containing information—such as the QuickDraw graphics system global variables, application global variables, and application parameters—accessed through the A5 register of a Motorola 68K processor. However, Mac OS 8 doesn't emulate the A5 world for applications compiled to run on PowerPC CPUs. Therefore any PowerPC-native code that makes assumptions about the organization of its A5 world won't work in Mac OS 8.

SUMMARY

Most code developed for Mac OS 8 uses instantiations of the per-process memory allocator to dynamically supply storage for temporary data. When code needs to share its temporary data, it can also use the system-wide memory allocator. Privileged code that can't tolerate page faults can also use the nonpageable-memory allocator. In addition to using these system-supplied memory allocators, developers with special needs can create and use custom memory allocators. Memory allocators automatically increase the amount of storage necessary to satisfy the needs of their clients, and these allocators support preemptive scheduling to make the system as a whole perform more efficiently.

Mac OS 8 offers the Memory Manager as a compatibility service for System 7 applications. However, applications perform much faster when using a Mac OS 8 memory allocator instead of the Memory Manager. Adherence to the forward-compatible memory guidelines allows a System 7 application to invoke the per-process memory allocator even when the application uses the Memory Manager. Nevertheless, compared to the Memory Manager, the Dynamic Storage-Allocation Service offers an easier-to-use, pointer-based programming model.

PLANNING A PRODUCT FOR MAC OS 8

If you're a developer, you can take the following steps to prepare products that take advantage of the Dynamic Memory-Allocation Service:

1. Remove all assumptions in your code about the physical layout of memory, such as the relative positions of heaps, stacks, and other memory areas used for storing data.
2. If you develop OpenDoc part editors for System 7, remove any calls to the System 7 Memory Manager from your code and rely on the OpenDoc memory management protocol exclusively. Your products will then be forward compatible with the Mac OS 8 per-process memory allocator.
3. Consider whether your code has unusual storage-allocation needs that might by optimized by supplying your own custom memory allocator.
4. If you want to migrate the code base for an existing System 7 application to Mac OS 8, begin changing your code so that it adheres to these forward-compatible memory guidelines from Apple Computer: (1) don't dispose of pointers and handles allocated indirectly by the Toolbox; (2) don't access handles, pointers, or heap zones outside the application heap or system heap; (3) don't allow application plug-ins to call Memory Manager routines—instead, ensure that they call the application to perform their memory management; (4) don't allocate memory from the system heap or from temporary memory; (5) remove all calls to the following Memory Manager routines: InitApplZone, SetApplBase, InitZone, GetApplLimit, SetApplLimit, MaxApplZone, MoreMasters, NewHandleSys, NewHandleSysClear, NewEmptyHandleSys, HandleZone, RecoverHandle, NewPtrSys, NewPtrSysClear, PtrZone, FreeMemo, MaxMem, CompactMem, ReservMem, PurgeMem, TopMem, GrowZoneProcs, and PurgeProcs.

8 The Run-Time Environment

Computer users want efficiency and flexibility in their systems, and the Mac OS 8 run-time environment is designed to provide both: first, by efficiently sharing code to reduce memory requirements and, second, by supporting flexible mechanisms for updating software. The **run-time environment** is the set of conventions that arbitrate how software is generated into executable code, how code is mapped into memory, and, at execution time, where data is stored, how data is addressed, and how functions call one another. The run-time environment isn't visible to users. However, users are generally aware of system and application memory requirements and of the relative ease or difficulty of upgrading applications and the operating system. The run-time environment greatly influences memory use and the ease of upgrading.

In the Mac OS 8 run-time environment, shared libraries make efficient use of memory. A shared library allows several programs to use code that's mapped to a single location in memory. For example, because the Mac OS 8 File Manager is implemented in a shared library, all tasks that open and close files share a single memory-mapped copy of File Manager code.

Shared libraries foster extensibility, which simplifies software upgrades and enhancements. Because shared libraries are dynamically linked, Apple or another developer can, for example, enhance a portion of the Mac OS 8 human interface by shipping an updated version of a shared library that's easy to install and is reliably integrated with the rest of the system.

This chapter focuses on the architecture of the run-time environment. It explains how the run-time environment makes efficient use of memory and how it supports applications developed for System 7. Chapter 9 describes how the run-time environment simplifies software extensibility.

KEY TERMS AND CONCEPTS

- A **code fragment** is a block of executable code and its static data. Code fragments are created by programming tools at generation time. Executable code generated specifically for Mac OS 8 is made up entirely of code fragments. The generation-time creation of code fragments and their execution-time preparation and use form the basis of the Mac OS 8 run-time environment.
- **Generation time** is the time during which executable code is created from source code using such program-development tools as a compiler and linker. For example, a developer creates a program at generation time.
- **Link time** is the point during generation time at which a linker binds object code with imported libraries to create executable code.
- **Execution time** is the general span of time during which programs run on a computer. During this time, code implemented in shared libraries is prepared for use by the applications that call these libraries.
- A **library** is computer code stored in a file or set of files for use by other code. A library provides building blocks of code for commonly needed operations. In the Mac OS 8 run-time environment, all libraries are implemented as shared libraries based on code fragments; the Mac OS 8 run-time environment doesn't allow the creation of libraries that aren't shareable (or at least potentially shareable) by more than one program.
- A **shared library** is a code fragment exporting a set of routines and data, or a data-only fragment exporting data, that can be used by other programs. Because they are prepared for use dynamically—that is, at code-execution time instead of at generation time—shared libraries are also called **dynamically linked libraries**. There are two types of shared libraries: import libraries and plug-ins.
- An **import library** is a shared library automatically prepared by the Code Fragment Manager for use by a program at launch time. (**Launch time** is the period during which the operating system builds the process for a program that is starting up.) The Code Fragment Manager prepares an import library to resolve imported symbols in program code that weren't resolved at link time.

- An **imported symbol** is a name used in a code fragment. The imported symbol references a discrete element of code or data in an import library.
- A **plug-in** is a shared library dynamically located and prepared for use by another code fragment when the code fragment explicitly calls Code Fragment Manager functions.
- **Static data** consists of variables and other data for which memory is allocated once so that the data persists between calls to a particular code fragment.
- A **data-only fragment** is a block of static data created by programming tools. A data-only fragment is occasionally used to implement a shared library.
- The **Code Fragment Manager** is the operating system service that prepares programs and import libraries for execution.

MAC OS HERITAGE

The Term *Fragment*

When developing the run-time environment for its first PowerPC-based computers, Apple Computer chose the term *code fragment* to avoid confusion with other terms (such as segment, object, component, and module) that Apple had already used to describe executable code. The term *code fragment* isn't intended to suggest that a block of code is in any way small, detached, or incomplete. Code fragments can be of virtually any size, and they are complete, executable entities. Some fragments, called *data-only fragments* in this book, contain no code; instead they contain data used by other code fragments.

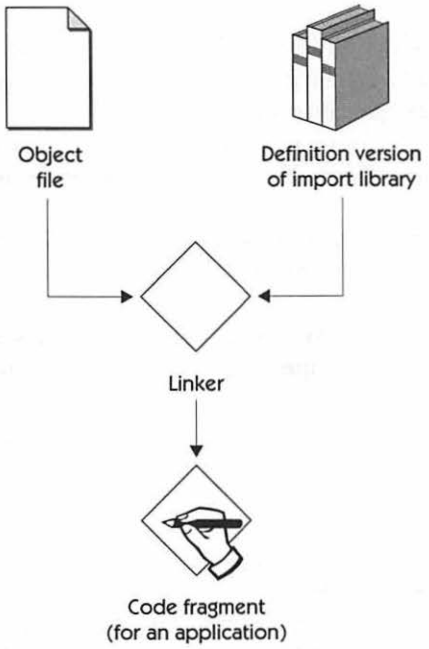
MAJOR POINTS OF INTEREST

A **compiler** is a tool that converts source code written in a high-level language like C into object files containing instructions in machine language. A **linker** is a tool that creates executable files by binding object files with libraries.

Conventions established by the run-time environment arbitrate how software is generated into executable code, how code is loaded into memory, where data is stored and how it is addressed, and how functions call other functions. For Mac OS 8, these conventions are implemented by software development systems (such as the compilers and linkers that generate executable code), the Code Fragment Manager (which manages the preparation of executable code), the microkernel (which maps code and data into memory and schedules code for execution), and the CPU (which executes code).

All code compiled specifically for Mac OS 8, whether for cooperative programs, server programs, device drivers, or the operating system itself, is linked into packages known as code fragments. A developer creates a code fragment at generation time by compiling source code into object code and linking this

FIGURE 8.1 The creation of a code fragment at generation time


Object file
Definition version of import library
Linker
Code fragment (for an application)

with other code fragments called **import libraries.** As Figure 8.1 shows, a linker produces a code fragment from an object file and an import library. (An import library can be considered to have two versions: a definition version used at generation time and an implementation version used at execution time.)

Program source code typically calls routines defined in import libraries. For example, a cooperative program usually includes calls that draw windows, and these calls are defined in an import library. The code that implements these calls, however, isn't in the code fragment for the cooperative program. Instead of placing code or data from an import library directly into the program's code fragment at link time, the linker places imported symbols into the code fragment. An imported symbol references code or data contained in an import library.

An import library is also a fragment. The code fragment for an import library differs from the code fragment for a program in that the former isn't an independently executable entity. Instead, it contains code that can be called from other code fragments, such as those for programs and those for other shared libraries.

When a program is launched, the Code Fragment Manager prepares the program's code fragment for execution. Part of this preparation involves matching the fragment's imported symbols with code or data supplied by the implementation versions of import libraries. For example, code supplied by the Human Interface Toolbox to implement windows, controls, and menus becomes available to an application only after the user launches the application. Then when the application calls a function to draw a window, code for the Human Interface Toolbox actually draws the window.

At any time a program is running, it can call the Code Fragment Manager to prepare code fragments known as **plug-ins**. The program and its plug-ins adhere to a programming interface allowing the program to call its plug-ins at execution time. For example, the developer of a graphics application might define a programming interface for use by plug-in developers. Developers of graphics utilities could then provide plug-ins that add new capabilities to the graphics application. One developer might create a plug-in that allows graphic artists to alter light sources in illustrations, and another developer might supply a plug-in that allows artists to perform custom color blending. To use a plug-in, a program or other code fragment must explicitly call the Code Fragment Manager at execution time.

Both plug-ins and import libraries are shared libraries. Multiple code fragments can simultaneously use a single copy of a shared library. For instance, when two or more applications use the QuickDraw 3D graphics system library, only one copy of its code is mapped into a system-wide memory area. Software developers can take advantage of shared code to reduce memory requirements for their own products.

Because they are prepared for use dynamically, that is, at execution time instead of at generation time, shared libraries are also called **dynamically linked libraries**. The dynamic, execution-time preparation of shared libraries further reduces memory requirements. The operating system allocates memory for a shared library only when it's referenced by another code fragment. In particular, an import library is mapped into logical memory only at launch time for a program that references that library, and a plug-in is mapped into logical memory only when a program explicitly uses Code Fragment Manager functions during execution time. The import library for QuickDraw 3D, for example, is mapped into memory only when an application that references this library is launched. All subsequently launched programs using QuickDraw 3D share the same memory-mapped copy of QuickDraw 3D code. When the last application to use this library quits, the memory area allocated to QuickDraw 3D code is released.

The virtual memory system (as you saw in Chapter 6) ensures that code doesn't occupy any space in physical memory until the CPU needs that code. Code for a shared library, therefore, is loaded into physical memory only when called by another code fragment and then released from physical memory when no longer needed.

Dynamic linking separates program code from shared library code until execution time. This separation makes it easier for developers to update software because shared libraries can be distributed independently of the programs that use them. For example, Apple Computer can distribute an updated version of the shared library containing routines for drawing and managing onscreen controls. At launch time, every cooperative program would become dynamically linked to the new code in the updated library. In this way, every enhancement Apple makes to controls would be reflected in every program that uses controls.

The run-time environment for Mac OS 8 is a continuing evolution of the run-time environments for several previous versions of System 7. The Mac OS 8 run-time environment supports applications created for System 7, although Mac OS 8 emphasizes the performance of fragment-based programs over System 7 applications compiled for the 68K family of Motorola processors.

MAC OS HERITAGE

The PowerPC Run-Time Environment for System 7.1.2

The Mac OS 8 run-time environment is an evolution of the one introduced with System 7.1.2 for PowerPC-based Mac OS–compatible computers. Fostering forward application compatibility, this run-time environment unifies mechanisms for loading and executing code so that developers' binary files automatically gain new capabilities as Apple Computer continues releasing new versions of the Mac OS.

FRAGMENTS

In Mac OS 8, as in System 7, fragments are created at generation time. Every program consists of one or more code fragments. A code fragment typically consists of

A **variable** is a named storage location for a modifiable value. A **global variable** is a named storage location for a modifiable value that can be referenced outside the local scope of code statements using that variable.

- One or more **code sections**, which contain binary code—that is, compiled instructions for the CPU to execute. The Code Fragment Manager maps the code sections of all fragments into read-only areas of system-wide memory.
- A **data section**, which contains the static data—including pointers to functions and pointers to global variables—used by code in the code section of the fragment. The Code Fragment Manager typically uses a per-process memory allocator for storing the data section, but the developer of a fragment can also direct the Code Fragment Manager to use the system-wide memory allocator instead.

A **memory allocator** is a plug-in used by client code for dynamic storage allocation. Mac OS 8 instantiates a **per-process memory allocator** for every process. The **system-wide memory allocator** allows any code to allocate storage from system-wide memory areas.

Most shared libraries contain code. However, some shared libraries, called data-only fragments, contain no code. A data-only fragment contains a data section but no code section. The data in a data-only fragment can be used by the code in other code fragments. Data-only fragments, as you'll see later in this chapter, are occasionally used for sharing system-wide static data.

Because all fragments are potentially shareable (although not all are actually shared), the terms *fragments* and *shared libraries* are often used interchangeably. While all shared libraries are fragments in Mac OS 8, however, not all fragments are shared libraries. In general, a shared library supplies code or data for use by other code fragments during execution time.

SHARED LIBRARIES

All code is potentially shareable as a library in Mac OS 8. A developer who wants to create a library of shareable routines generates a code fragment from source code for the routines, directing the linker to label that code fragment as a shared library. It is up to the developers of client code to import and use that shared library. One developer, for instance, can create a shared library consisting of a spell-checking utility. By importing that shared library into their programs, other developers can include the spell-checking utility in their products, too.

There are two types of shared libraries:

- import libraries, which the Code Fragment Manager automatically prepares for use by client code fragments when they are launched
- plug-ins, which the Code Fragment Manager prepares for execution only when client code fragments explicitly call the Code Fragment Manager

Import Libraries

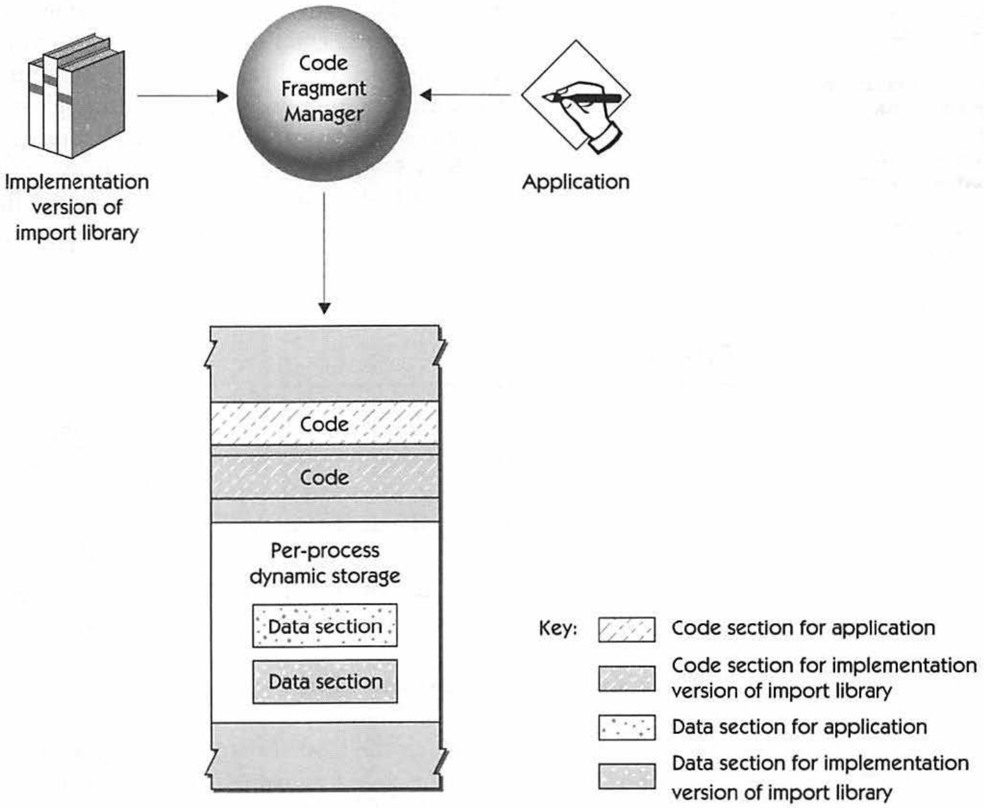
An import library may be considered to have two versions:

- its **definition version**, which is used by client code at generation time
- its **implementation version**, which is used by client code at execution time

The definition version defines the external programming interface and data format of the library. The implementation version contains the actual data and executable code supplied by the library.

A code fragment, such as one for an application, references the definition version of an import library at generation time. A programmer begins by com-

FIGURE 8.2 Preparing an import library for use by a client code fragment at launch time


Code Fragment Manager
Implementation version of import library
Application
Code
Code
Per-process dynamic storage
Data section
Data section
Key:
Code section for application
Code section for implementation version of import library
Data section for application
Data section for implementation version of import library

piling source code into an object file. As shown in Figure 8.1 on page 112, the programmer then passes this object file, along with the import library, to a linker. The linker creates a code fragment, which contains imported symbols to the functions and variables defined in the definition version of the import library.

Figure 8.2 illustrates the launch-time preparation of the implementation version of an import library. When an application is launched, the Code Fragment Manager prepares for execution the application code fragment and the code fragments for all import libraries referenced by the application. As this figure illustrates, the code for an application code fragment is mapped into one memory area, and the code for the import library is mapped into another memory area. As for all code, these memory areas are shared system wide and have read-only access permissions.

To **instantiate** something is to create an instance of it at execution time.

The Code Fragment Manager typically instantiates the data sections of the application's code fragment and the import library's code fragment by using the application's per-process memory allocator (described in Chapter 7). The Code Fragment Manager then resolves the imported symbols in the application code fragment by converting their symbolic addresses to the actual memory addresses of routines and data supplied by the import library.

All Mac OS 8 services are packaged as import libraries. For example, Human Interface Toolbox services are packaged as import libraries. At generation time, the developer of a cooperative program links the object code for that program with definitions versions of the Human Interface Toolbox libraries. Later, when the user launches that program, the Code Fragment Manager automatically prepares the code within these libraries for use by the cooperative program.

This example further illustrates the two major benefits of a shared library: memory conservation and ease of software updates. Because code for the Human Interface Toolbox is mapped only once into logical memory for use by all programs, memory requirements across the system are reduced. Thus, all programs that are linked at their generation times with the Human Interface Toolbox—as well as any other Mac OS 8–supplied shared libraries—automatically take advantage of code sharing. The dynamic preparation of shared libraries at execution time further reduces memory requirements by allocating dynamic storage for a shared library only when it's referenced by another code fragment. And because the shared library code becomes available to programs only at execution time, Apple Computer can distribute library updates to users and thereby automatically provide all applications with the most up-to-date versions of human interface features.

In addition to using Mac OS 8–supplied libraries, software developers typically provide their own import libraries. For example, the developer of a scientific simulation application might define and use an import library of specialized math functions. If the developer revises these functions to provide better performance, enhanced capabilities, or greater reliability, the developer can easily upgrade users by distributing updated versions of this library, as you'll see in Chapter 9.

Plug-Ins

A developer can create shared libraries in the form of plug-ins. The Code Fragment Manager doesn't automatically prepare plug-ins for use by a program when it's launched. Instead, at any point during execution time, a program calls the Code Fragment Manager to prepare a plug-in for the program's use.

A plug-in is prepared for use during execution time only when requested by a client program. A plug-in adheres to a programming interface defined by the developer of a client program. For example, suppose a developer of a digital-video editing program defines a programming interface for plug-ins that perform special editing effects. Other developers might then produce and market these plug-ins, and the user might purchase these plug-ins separately from the digital-video editing program. When executing, the program would display, perhaps in a menu, the special editing effects provided by all installed plug-ins. If the user chose a special editing effect, the program would call the Code Fragment Manager to prepare the plug-in so that the program could use the plug-in's code.

Like an import library, the code for a plug-in is mapped once into a system-wide memory area, allowing multiple programs to use that code. Also, a developer can update a program simply by distributing an enhanced version of a plug-in for the program. Because plug-ins aren't linked with a program during generation time, plug-ins are in many ways more flexible than import libraries.

For example, if a developer created a video-effects editor as an import library, that editor would be prepared for use by a program and mapped into memory when the program was launched, but if the video-effects editor were implemented as a plug-in, it would be prepared and mapped into memory only when the user chooses to use it. Further, if the video-effects editor were implemented as a plug-in, its developer could make an audio-effects editor and an animation-effects editor available as additional plug-ins. Because the Code Fragment Manager doesn't prepare plug-ins at application launch time, an application would require less memory and launch more quickly if its special-effects editors were implemented as a family of plug-ins rather than as a single large import library.

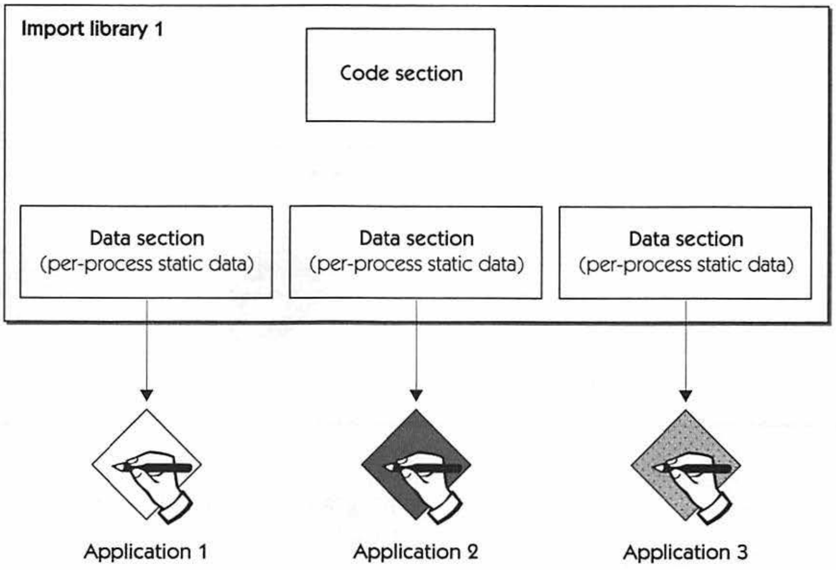
Perhaps the ultimate plug-in architecture is offered by OpenDoc. When a user creates a new part in an OpenDoc document, the Code Fragment Manager dynamically prepares for execution the part editor associated with that part. For example, a user can add a text-editing part to a compound document any time the user works with that document. As with any plug-in, the code for that part editor is mapped only once into a system-wide memory area. If the user were to add that part to several documents, all instances of its part editor would use the same code, thereby reducing memory requirements across the system.

COMPATIBILITY NOTES

The Apple Shared Library Manager

The Apple Shared Library Manager (ASLM), a shared library technology available on some earlier versions of the Mac OS, isn't supported in Mac OS 8. Instead, it has been superseded by the Code Fragment Manager and SOMobjects for Mac OS which, as you'll read in Chapter 9, is implemented on top of the Code Fragment Manager. SOMobjects for Mac OS has the particular benefit of being an industry-standard solution that's both compiler-neutral and language-neutral. This solution allows object-oriented libraries to be called from code written in any development environment. Another important reason that Apple Computer

FIGURE 8.3 Separate instantiations of the data section for an import library


Import library 1
Code section
Data section (per-process static data)
Data section (per-process static data)
Data section (per-process static data)
Application 1
Application 2
Application 3

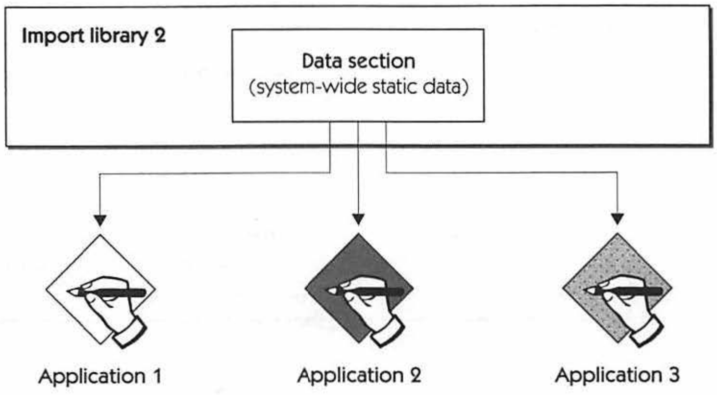
has chosen the Code Fragment Manager and SOMobjects over ASLM is to ensure greater operating system synergy and consistency with OpenDoc, an important client of this shared library solution.

STATIC DATA INSTANTIATION

The Code Fragment Manager always maps a single copy of the code of a shared library into a system-wide memory area. However, the Code Fragment Manager can instantiate the data section of a shared library using either a per-process memory allocator, so that the tasks of that process have their own private instance of the data, or the system-wide memory allocator, so that all tasks in the system have a common instance of the data.

For an import library supplied as part of the operating system, the Code Fragment Manager typically instantiates a separate copy of the library's data section with the per-process memory allocator of every program using that library. Figure 8.3 illustrates three applications importing the same shared library. These applications all share the library's code section, but each application gets its own copy of the library's data section. The Code Fragment Manager, for instance, instantiates a separate copy of the data section of the

FIGURE 8.4 A single copy of the data section for an import library



file system for every program using that library. The file system uses separate data sections, one for each program, to track what files have been opened by which programs.

This instantiation method is called **per-process instantiation,** and the data instantiated this way is called **per-process static data.** This data is available to all tasks in the process for that program, and this data is generally available *only* to the tasks for that program.

Code Fragment Manager documentation from Apple Computer sometimes refers to per-process instantiation as **per-context instantiation.**

Developers typically create shared libraries that use per-process instantiation. For example, it's easier to convert a statically linked library into a shared library by implementing it as one using per-process static data.

The Code Fragment Manager can also perform **system-wide instantiation,** in which case the Code Fragment Manager supplies an instance of a shared library's data section using the system-wide memory allocator; the data instantiated this way is called **system-wide static data.** System-wide instantiation allows all tasks in the system to use the same instance of the static data. Figure 8.4 illustrates several applications sharing the same instance of the data section of a shared library.

Documentation from Apple Computer sometimes refers to system-wide instantiation as **global instantiation.**

Only a small number of shared libraries supplied by the operating system are instantiated with system-wide static data. One such library is supplied for the Timing Services. All programs and shared libraries that use the Timing Services share the same data so that they can rely on system-wide timing values.

Developers can also create shared libraries that employ system-wide static data, but the compiler won't allow import libraries using system-wide static data to import libraries using per-process static data. Otherwise, if a code fragment with system-wide static data were to inadvertently import another shared library—for instance, by calling one of its routines—the per-process data used by the imported library would have valid addresses for only one process. The fragment with system-wide data would then be reliable for only one process. Developers work around this situation by keeping system-wide static data in data-only fragments—fragments that contain no imported symbols. A shared library containing no imported symbols can't inadvertently import routines or data contained in other shared libraries.

It's also possible to instantiate a new copy of a plug-in's static data every time a program calls the Code Fragment Manager to prepare the plug-in for use. This type of instantiation, which uses a per-process memory allocator for allocating storage for the data, is called **private-copy instantiation.** All private-copy instantiations of a plug-in continue to use the same copy of the plug-in code, but each instantiation has its own copy of plug-in data. A communications application might use a plug-in to implement a tool for connecting to a serial port, for example. After requesting the Code Fragment Manager to perform multiple instantiations of the tool, the application can use the tool to connect to two or more serial ports simultaneously by maintaining separate copies of the tool's static data.

Private-copy instantiation is sometimes called **per-load instantiation** in documentation from Apple Computer.

COMPATIBILITY NOTES

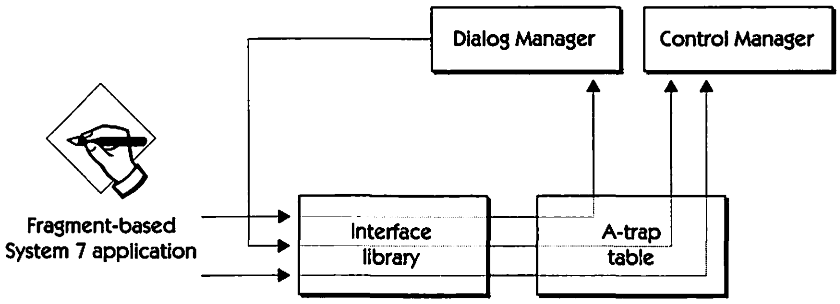
Static Data Instantiation in System 7

The System 7 version of the Code Fragment Manager instantiates per-process static data in application heaps and all system-wide static data in the system heap. Whereas a shared library supplied by Mac OS 8 is typically instantiated with per-process static data, many shared libraries supplied by System 7 are instantiated with system-wide static data. The System 7 approach allows one program to easily change many of the operating system's global variables, which are often used by other programs. Many system extensions in System 7 do this in very clever and useful ways. However, the patching of system-supplied global variables occasionally leads to system instability for applications that rely on these variables remaining unaltered by other programs. The predominance of per-process instantiation in Mac OS 8 increases system reliability by preventing other code from changing global variables supplied by the operating system. For this and other reasons, System 7 extensions don't work in Mac OS 8. As you'll read in Chapter 9, Mac OS 8 provides a variety of other mechanisms that developers can use to reliably extend or modify the operating system.

RUN-TIME–ENVIRONMENT SUPPORT FOR SYSTEM 7 APPLICATIONS

The Mac OS 8 run-time environment supports code-fragment–based software generated for System 7. The Mac OS 8 run-time environment also supports System 7 software based on the use of the A-trap table—a central part of the

FIGURE 8.5 Access to system services in System 7



original Macintosh run-time environment—by running this software under emulation.

MAC OS HERITAGE

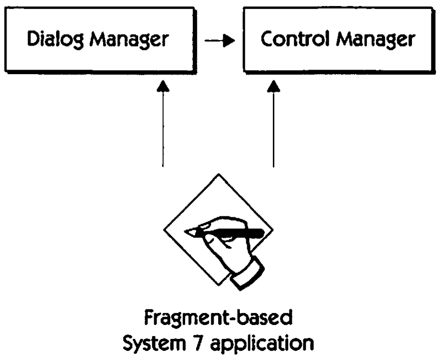
The A-Trap Table and A-Traps

An **A-trap table** contains a list of A-traps—that is, entry points to Mac OS routines called by code generated to execute on the 68K CPUs. An **A-trap** is a compiled instruction that is unimplemented by the Motorola 68K family of microprocessors. The first 4 bits of such an instruction have a hexadecimal value of A. For applications compiled to run on 68K-based computers, these instructions invoke routines implemented by the Mac OS. For example, the call to create a window has a hexadecimal value of A913, which causes the CPU to generate an exception. The exception invokes an exception handler provided by the Mac OS, allowing the operating system to "trap" the routine and respond by creating a window.

System 7 and the A-Trap Table

The code-fragment–based run-time environment of the Mac OS began with the introduction of System 7.1.2 for PowerPC-based computers. In this and all versions of the Mac OS before Mac OS 8, every program accesses operating system services through the A-trap table. In System 7, A-trap–based software uses the trap table directly; code-fragment–based software calls an interface library, which in turn uses the trap table. (All native PowerPC software developed for System 7 is code-fragment based.) Figure 8.5 shows calls from a code-fragment–based application in System 7 to two shared libraries, one for the Dialog Manager and one for the Control Manager, going through the interface library and trap table. It shows the same process for the Dialog Manager using the shared library of the Control Manager.

FIGURE 8.6 Access to system services in Mac OS 8



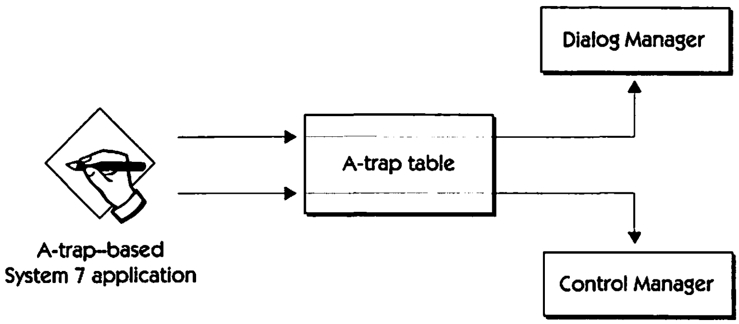
The System 7 run-time environment for PowerPC computers uses the A-trap table to maximize the compatibility and performance of A-trap–based programs, which predominated at the introduction of PowerPC-based Macintosh computers. Since then, Apple Computer has been quickly shifting its product line to PowerPC-based Macintosh computers, and other manufacturers have begun licensing the Mac OS to run on their own PowerPC-based computers; development of A-trap–based programs has subsequently decreased. Instead, developers have focused on delivering code-fragment–based programs that take advantage of PowerPC performance. Even for developers continuing to create products for computers using the Motorola 68K family of microprocessors, Apple Computer has provided a code-fragment–based run-time environment for the Motorola 68K platform.

As users over the last several years have migrated to PowerPC-based computers and code-fragment–based software, the performance of A-trap–based software has decreased in importance. With the introduction of Mac OS 8, Apple Computer optimizes the performance of code-fragment–based software over the performance of A-trap–based software.

Code-Fragment–Based Software in Mac OS 8

Whereas all code-fragment–based calling conventions in Mac OS 8 remain consistent with those of System 7, all code-fragment–based software gains direct access to the shared libraries supplied by Mac OS 8. Because there is no longer a circuitous route through an interface library and a trap table, the performance of code-fragment–based software is increased, even for code-fragment–based software written for System 7. Shared libraries supplied by the operating system also access each other directly, one to another. Figure 8.6 shows a code-fragment–based System 7 application gaining direct access in

FIGURE 8.7 Mac OS 8 support for A-trap–based software



Mac OS 8 to the shared libraries of the Dialog Manager and the Control Manager, which are supplied for System 7 application compatibility. The figure also shows the shared library for the Dialog Manager gaining direct access to the shared library for the Control Manager.

COMPATIBILITY NOTES

Code Fragments for PowerPC-Based and 68K-Based Computers

Code fragments generated for 68K-based computers won't run on PowerPC computers, nor will code fragments generated to run on PowerPC-based computers run on 68K-based computers. For code fragments to run on both types of computers, developers must compile them twice—once for each type of computer.

Mac OS 8 Support for A-Trap–Based Software

The Mac OS 8 run-time environment continues to support A-trap–based software, although this software is no longer the focus of the run-time environment as in System 7. As illustrated in Figure 8.7, this environment supplies a trap table that references the shared libraries supplied by Mac OS 8. For every process that the operating system creates for A-trap–based software, the operating system creates a separate trap table. In other words, every A-trap–based application gets its own copy of the trap table.

SUMMARY

The Mac OS 8 run-time environment is based on the generation-time and execution-time preparation of code fragments. Code generated specifically for Mac OS 8 consists entirely of code fragments.

Code fragments can be packaged in shared libraries. A shared library allows multiple other code fragments to use a single copy of its memory-mapped code, thus reducing memory requirements. Moreover, shared libraries are dynamically mapped into memory at execution time. This feature also makes it easier to update users with enhanced software, as you'll see in Chapter 9.

Mac OS 8 services are made available to developers through shared libraries. Software developers can implement features for their own products in shared libraries. There are two types of shared libraries: import libraries and plug-ins. Import libraries are automatically prepared for execution by the Code Fragment Manager to resolve imported symbols in the code fragments of programs being launched. Plug-ins are prepared for execution by the Code Fragment Manager only at the request of client programs.

The Mac OS 8 run-time environment is optimized for code-fragment–based software, although Mac OS 8 continues to support System 7 applications generated to run on 68K-based computers.

PLANNING A PRODUCT FOR MAC OS 8

If you're a developer, you can take the following steps to prepare products that take advantage of the Mac OS 8 run-time environment.

1. Generate your existing System 7 products as code fragments—in particular, if you haven't already done so, update your products by making them PowerPC-native for System 7.
2. If your existing System 7 product uses the Code Fragment Manager to create system-wide static data, place this data in an import library implemented as a data-only fragment, and reference no other shared libraries from this data-only fragment.
3. Make no assumptions in your code about the layout of memory, such as the locations of your application global variables.

9 Software Extensibility

To support technological advances and the customization needs of developers, Mac OS 8 incorporates product extensibility. From its lowest hardware-related services to its most abstract human interface features, Mac OS 8 provides mechanisms for enhancing the entire operating system. This extensibility allows Apple Computer and Mac OS 8 developers to tailor the operating system to fit present needs and expand it in the future to incorporate technological innovations. These operating system mechanisms also make it easier for developers to build extensibility into their own products. Users, in turn, benefit by gaining rapidly developed product enhancements.

Extensibility is the ability of software to be enhanced with new capabilities without breaking those it already supports. Mac OS 8 incorporates several mechanisms by which developers can extend the operating system as well as their own products. OpenDoc offers the most flexible mechanism for delivering extensible application-level software. Shared libraries, from which OpenDoc part editors are built, provide a lower-level mechanism for Apple and developers to update and enhance any type of software. SOMobjects for Mac OS provides an object-oriented approach for using shared libraries to extend software.

Server programs are yet another mechanism for software extensibility, allowing developers to extend the capabilities of the operating system and their own products. And developers who have a pressing need to modify individual routines in shared libraries can use the Patch Manager to do so reliably.

KEY TERMS AND CONCEPTS

- OpenDoc is a multiplatform technology, implemented as a set of shared libraries, that facilitates the construction and sharing of compound documents. Each compound document can consist of multiple user-selected parts, which create, contain, and display information.
- A **shared library** is a code fragment exporting a set of routines or static data that can be called by multiple programs. Shared libraries are prepared dynamically for program use. An **import library** is a shared library automatically prepared by the Code Fragment Manager at program launch time. The Code Fragment Manager prepares an import library to resolve imported symbols in program code that weren't resolved at link time. By comparison, a **plug-in** is a shared library dynamically located and prepared for use by another code fragment when the code fragment explicitly calls Code Fragment Manager functions at any point during execution time.
- **SOMobjects for the Mac OS** is the Apple Computer implementation of the System Object Model (SOM), an industry standard architecture licensed by IBM for the development and packaging of object-oriented software. Developers who desire to build object-oriented extensibility into their programs can use SOMobjects for the Mac OS in their own products. Apple uses SOMobjects for the Mac OS to implement OpenDoc shared libraries and to provide object-oriented extensibility in Mac OS 8. The human interface elements in Mac OS 8 and several other key operating system features, for example, are implemented with SOMobjects for the Mac OS, allowing Apple and other developers to modify these features easily while maintaining future compatibility between applications and the operating system.
- A **server program** operates on data within its own protected address space. Server programs typically provide services to other programs along a client/server model.
- The **Patch Manager** is service that helps developers modify routines supplied in import libraries.

MAJOR POINTS OF INTEREST

Extensibility mechanisms allow Apple and other developers to enhance Mac OS 8. These mechanisms also allow developers to quickly adopt new advances in their own applications.

For users and developers alike, OpenDoc provides the greatest degree of flexibility in creating application-level software that integrates seamlessly with users' personally configured work environments. A user can buy or create a compound document containing various part editors; part editors, in turn, are responsible for manipulating specific types of content within the parts of the document. A compound document in the OpenDoc environment acts like a shell to hold various parts. The user can add or remove a part just by dragging it in or out of a compound document. As the user works with a part, the code for that part editor runs and manipulates the data within the part. Every part editor is implemented as a shared library, which is mapped into a system-wide memory area for use by all compound documents containing that part editor.

The prevalence of shared libraries and server programs throughout the system simplifies the future improvement of the Mac OS 8 platform. For example, if Apple chooses to enhance the appearance of onscreen controls in a later release of the Mac OS, Apple needs only replace a shared library instead of releasing a new version of the entire operating system. Software developers can take advantage of this extensibility in their own products, too. For example, a developer could add new features to a spelling checker implemented as a shared library and ship a revised version of this library to customers without requiring them to update the word processors, e-mail editors, and page-layout programs that use the spelling checker. Such extensibility makes it very easy for users to keep their software up to date.

Mac OS 8 also supports extensibility through object-oriented programming techniques. The Mac OS 8 adoption of SOMobjects for Mac OS helps developers to extend their own software—as when they add new capabilities to existing products—or to extend Mac OS 8 in new ways—as when developers customize standard Mac OS 8 menus or controls for unique requirements. SOMobjects for Mac OS is built on top of the Code Fragment Manager, and thus class libraries are implemented as shared libraries.

A developer needs to use the object-oriented features of the SOM classes provided by Mac OS 8 only to modify human interface features or other SOM-implemented features of the operating system. For most developers, such modifications are unnecessary. Instead, using their preferred programming languages, most developers simply call Mac OS 8–supplied functions, which in turn manipulate objects instantiated from unmodified SOM classes. Because of the language-neutral nature of the System Object Model, developers can use procedural as well as object-oriented languages. Only a minority of developers will wish to alter the default behavior or appearance of operating-system elements derived from SOM classes. For these developers, SOMobjects for the Mac OS provides a reliable way to modify portions of the operating system while maintaining forward compatibility with future versions of Mac OS 8.

To extend the capabilities of their own products or the operating system, developers can create server programs. For example, to satisfy the needs of a suite of cooperative programs—such as a scientific simulation program, an engineering design program, and a three-dimensional rendering program—a developer can supply a single server program that performs data-intensive statistical simulations. To incorporate future improvements to the statistical simulation capabilities of these cooperative programs, the developer needs only supply customers with a revised version of the server program.

Finally, to extend individual routines supplied in import libraries—such as those implementing Mac OS 8 programming interfaces—developers can use the Patch Manager.

This chapter describes the software extensibility mechanisms provided by Mac OS 8. The modular design of Mac OS 8 also makes it easier for developers to extend and differentiate the operating system for different hardware platforms. For example, a developer might integrate a specialized video capture device into the I/O system while implementing a volume format optimized for video data. Chapter 11 describes the modular nature of the I/O system, and Chapter 10 describes the volume format extensibility supplied by the file system.

COMPATIBILITY NOTES

Mac OS 8 Incompatibility with System 7 Extensions

To effect system-wide changes for all applications in System 7, developers often modify operating system features by creating system extensions (that is, files of type 'INIT') and patching the system-wide A-trap table. To improve system stability, however, Mac OS 8 doesn't support 'INIT' files. Moreover, the A-trap table mechanism is supported in Mac OS 8 only for backward compatibility with System 7 applications generated for 68K-based CPUs.

As explained in the rest of this chapter, however, Mac OS 8 provides other, more reliable mechanisms for developers to modify and extend operating system features.

EXTENDING SOFTWARE WITH OPENDOC

Parts are the portions of an OpenDoc document that contain content for view or manipulation by users. At execution time, **part editors** display part content, facilitate manipulation of the content, and provide a user interface for modifying that content.

The most flexible Mac OS 8 mechanism for extending application-level software is OpenDoc, because OpenDoc allows users to decide for themselves how to extend their work environments. OpenDoc technology centers around user-extensible documents. Users buy or create compound documents that incorporate multiple OpenDoc parts; these, in turn, supply users with the software capabilities they desire.

By supplying part editors based on their special areas of expertise, developers can help users extend the capabilities of their compound documents. For example, a developer of typesetting products can package an equation editor, a chart editor, and a typographical effects editor as OpenDoc part editors that users can purchase or that other developers might license.

A developer with a large and complex software product can separate product features into separate OpenDoc part editors. In this way, not only can the developer easily extend the product by offering new or improved part editors, but also the user can extend any compound document with this developer's part editors.

A developer of a large, standard application can make the application more extensible by allowing users to embed OpenDoc parts in the documents created with this application. The developer can also allow information to be linked between part editors and the application. Suppose, for example, that a personal finance application supports embedded OpenDoc parts in its documents. A user might then be able to extend a document created by that application with a part that periodically calls an online stock price quotation service. The application could then use that part to track the performance of the user's investments on an up-to-the-minute basis.

OpenDoc part editors are implemented as shared libraries prepared for use only when called from OpenDoc documents or applications that support the embedding of part editors. The use of shared libraries as a general extensibility mechanism is described next.

EXTENDING SOFTWARE THROUGH SHARED LIBRARIES

Because shared libraries are dynamically prepared at execution time, they allow users to update software without breaking other programs that rely on these libraries. As a result, Apple can supply updated shared libraries that extend Mac OS 8 features without requiring users to purchase new versions of their other software products. Similarly, shared libraries help developers update features in their own products. As you read in Chapter 8, there are two main categories of shared libraries, import libraries and plug-ins. Both can help developers extend software.

Using Import Libraries to Extend Software

When developers link the definition version of an import library into their code at generation time, they import symbols to the routines for that library. Library routines aren't available to programs until launch time, at which point the Code Fragment Manager associates imported symbols with the actual routines supplied in the implementation version of the import library. For this reason, developers or Apple Computer can replace the implementation version of an import library without changing the code that uses that library.

Because the vast majority of Mac OS 8 services are packaged as import libraries, Apple can easily extend these services in the future. If Apple wants to extend QuickDraw 3D in a later release of the Mac OS, for example, Apple can simply distribute the import library for QuickDraw 3D instead of releasing a new version of the entire operating system. Mac OS 8 developers wouldn't need to enhance their own products to incorporate a new version of QuickDraw 3D, because the operating system dynamically prepares the new version for use by these applications at the moment they're launched.

Similarly, other developers can divide their software products into pieces that lend themselves to periodic revisions and package these pieces as import libraries. For example, the developer of an animation application might define and use an import library of specialized image-rendering functions. If the developer revises these functions to provide better performance, enhanced capabilities, or greater reliability, the developer can easily provide upgrades by distributing updated versions of this library to users.

Using Plug-Ins to Extend Software

Developers can build extensibility into their software by creating shared libraries in the form of plug-ins. Unlike import libraries, which the Code Fragment Manager prepares automatically for programs at launch time, plug-ins are prepared for execution only at the request of client programs. Client applications define programming interfaces for their plug-ins. This approach gives developers a flexible mechanism for enhancing each other's products. For example, the developer of an e-mail application might define a programming interface to that application. Then other developers could provide plug-ins that add new capabilities to the application. One developer might create a plug-in allowing users to annotate e-mail messages with voice messages, and another developer might supply a plug-in that sends video clips along with e-mail messages.

HISTORY

Object-Oriented Programming

Whereas procedural programming involves the creation and organization of routines that pass control to one another, object-oriented programming focuses on building blocks of code called objects. An **object** is an execution-time structure that contains data and functions that operate on that data. An object is almost like its own small program in that it stores data and performs calculations on that data.

An object is always a particular instance of a more generic structure called a **class**, which can be used to create additional instances that constitute separate objects. A class is a template from which particular objects are created at execution time. For example, Mac OS 8 defines all controls—such as buttons, sliders, and checkboxes—in a class. Individual controls are instantiated as objects derived from this class.

Three key characteristics of object-oriented programming are encapsulation, inheritance, and polymorphism. **Encapsulation** is the packaging of an object's data and the functions that can act on it in order to protect the data from inappropriate changes. This protection is possible because only the object itself can change its data. To gain access to an object's data, a client must call that object's programming interface.

Inheritance is the transmission of the properties and behaviors from one class to another. Inheritance allows programmers to derive properties and behaviors of one object class from another. Inheritance makes it possible for programmers to extend software through **subclassing**. For example, a programmer can create a new subclass that inherits the behaviors of an existing superclass. The programmer can then add new behaviors and change inherited ones to supply more or different capabilities to the software.

Polymorphism is the ability to call objects of different classes with the same method. For example, a program might use the same method to draw objects defined by different classes.

Extending Software Through the System Object Model

In the opinion of many developers, object-oriented programming offers the most useful techniques for extending software. Shared libraries, as you have seen, provide a useful mechanism for developers to deliver software enhancements to users. However, C++ and various other object-oriented programming languages are often implemented in ways that make it difficult for developers to produce shareable libraries. To make object-oriented programming a viable approach to enhancing software through shared libraries, Mac OS 8 supports the System Object Model in its run-time environment.

The Apple implementation of the System Object Model, called SOMobjects for Mac OS, simplifies the work of developers who want to use object-oriented techniques to create software that's easily extended. SOMobjects for Mac OS was first released as the object-oriented underpinning for OpenDoc on System 7.5. SOMobjects for Mac OS allows Apple and other developers to design and package class libraries that support object-oriented approaches without facing many of the pitfalls associated with object-oriented programming languages—in particular, the inability to reuse binary code and various language incompatibilities between class libraries and the applications that use them.

The System Object Model in Mac OS 8

The **Code Fragment Manager**, described in Chapter 8, instantiates blocks of executable code and prepares them for execution.

SOMobjects for Mac OS is based on the Code Fragment Manager so that SOM classes are implemented as shared libraries. SOMobjects for Mac OS is used for

- standard user interface elements, such as windows, menus, controls, lists, and icons

- all Text Service Manager services, including interactive text services (like spelling checkers) and text input methods (like user keyboard activity)
- OpenDoc part editors

SOMobjects for Mac OS provides users with up-to-date features across applications, operating system releases, and application revisions. SOMobjects for Mac OS benefits developers and their users in these main areas:

- When Apple adds new features to subsequent versions of the Mac OS, users won't need to purchase revised applications to gain these features: applications will automatically inherit them. For example, if Apple refines a Text Service Manager service in a future version of the Mac OS, all Mac OS 8 applications using this service would automatically inherit the new refinements. With SOMobjects for Mac OS, Apple provides an easier way for users to keep their systems completely up to date.
- Developers can use SOMobjects for Mac OS to extend portions of Mac OS 8 for the benefit of their applications while retaining compatibility with future versions of the Mac OS. For example, an application can alter the default appearance of a control provided by Mac OS 8 in one release, and this modified behavior will continue to work in subsequent releases of the operating system.
- SOMobjects for Mac OS provides developers with an object-oriented approach to extensibility in their own code. For example, a developer can use SOM to implement product code for easier alteration and enhancement in subsequent releases.

HISTORY

The Object-Oriented Approach to Software Extensibility

As developers using object-oriented programming techniques know, classes facilitate the addition of features and capabilities to existing source code. For example, without changing any other code in a drawing program, a developer can override functions in the class for an object that draws itself as a two-dimensional black-and-white square so that the object instead draws itself as a three-dimensional color cube.

However, commercial object-oriented languages such as C++ suffer because they don't support the reuse of binary code—they support the reuse of source code only. For example, to make use of the object that can draw itself as a three-dimensional cube, a developer without the support of SOMobjects for the Mac OS might need to recompile the entire application. SOMobjects for the Mac OS allows this developer to provide users with this new three-dimensional drawing feature by simply distributing an updated import library instead of a completely recompiled application.

The Benefits of the System Object Model

The System Object Model provides developers with several key benefits. In particular, the System Object Model

- supports the key characteristics of object-oriented programming
- supports the release-to-release binary compatibility of class libraries
- provides compiler and language independence for the development and enhancement of code

The System Object Model supports encapsulation, inheritance, and polymorphism—the key characteristics of object-oriented programming. However, unlike libraries for other class types (such as C++ classes), SOM class libraries provide release-to-release binary compatibility. That is, a developer can replace individual class libraries in a software product without recompiling any of its other class libraries. In the future, for example, Apple might add new capabilities to the SOM class implementing windows. Apple can easily enhance Mac OS 8 windows by replacing a shared library, and—without being recompiled or relinked—applications will automatically inherit the window enhancements. Release-to-release binary compatibility also allows a subclass to be compatible with a new superclass. For example, if a developer creates an enhancement to Mac OS 8 windows, this enhancement will be compatible with future releases of Mac OS 8 even if Apple replaces the superclass for windows.

Whereas many compilers for object-oriented languages produce class libraries that are incompatible with different languages, the SOM approach to object-oriented programming provides compiler and language independence. Binary class libraries can be created in multiple languages—including procedural languages like C as well as object-oriented languages like C++. These libraries, in turn, can be used—and even subclassed—in different languages. For example, an Apple engineer can use one language to write a SOM class for Mac OS 8, and a developer can use another language to create a modified subclass of the Mac OS 8–supplied class. Better yet, an application written in another language can be linked with the shared library for the newly derived subclass—or with the original Mac OS 8-supplied class. Without the System Object Model, it's difficult for a programmer using one object-oriented language to produce class libraries for use by other object-oriented languages while also maintaining binary compatibility from one release of a product to the next.

The System Object Model isn't a complete implementation language or programming system. Instead, it complements existing languages that developers already use.

Using the SOM Classes Provided by Mac OS 8

The SOM classes provided by Mac OS 8 incorporate most of the features that developers have created for themselves in the past. For example, the Human Interface Toolbox provides such common (but previously nonstandard) features as floating windows, keyboard equivalents in menus, and tear-off menus. Therefore, the majority of developers have little need to subclass or recreate the SOM classes provided by the Human Interface Toolbox.

To use the standard appearance and behavior of windows in applications, developers at generation time call the programming interface provided by the Human Interface Toolbox with their preferred languages. Then developers can link their object files with the definition versions of Human Interface Toolbox libraries. At program launch time, the Code Fragment Manager automatically prepares, for application use, the implementation version of the import library responsible for drawing and managing windows.

MAC OS HERITAGE

Extensibility of Human Interface Elements in Macintosh System Software

All previous versions of the Macintosh operating system provide useful programming interfaces to help developers create and manage human interface elements. Despite the richness of these programming interfaces, many developers found the need to implement features that Apple didn't anticipate or supply. Examples include tear-off menus and tool palettes.

Many developer-supplied extensions to the human interface offered innovative new ease-of-use features. At the same time, this ad hoc reengineering of the operating system's human interface caused problems, including human interface inconsistencies among applications, additional programming and testing burdens for developers, system instability, and revision constraints for the operating system itself. By implementing many of its services in more easily extensible SOM classes, Mac OS 8 makes it much easier for developers to extend these services consistently and reliably.

Creating SOM Subclasses of Mac OS 8 Features

It's likely that only a small percentage of developers will wish to alter the default behavior or appearance of human interface elements or other operating system features implemented as SOM classes. For those who do, SOMobjects for Mac OS simplifies the job. For example, to create an entirely new control—say, a throw switch—a developer subclasses a Mac OS 8–supplied class for controls and then overrides its drawing functions.

The programming interface to an object class is described in the Interface Definition Language (IDL), a language resembling C++. The IDL files for all SOM classes in Mac OS 8 are available to developers who need to subclass these libraries. To subclass an existing class, a developer can use an IDL compiler to generate an implementation template file in any one of several supported programming languages, such as C and C++. The developer modifies this implementation template file and then uses a SOM compiler to create an updated object file, which the developer links with application code.

Using SOM Classes for Application Features

By packaging application features in SOM classes of their own, developers can use object-oriented techniques to extend these features in subsequent product revisions of their own code, too. For example, a developer might use SOMobjects for Mac OS to implement a plug-in to a graphics program. At its first release, this plug-in might allow users to alter the direction of a light source in illustrations. In subsequent releases, the developer can extend the product to support increasingly sophisticated effects—for instance, allowing users to alter the color and brightness of the light source.

Another benefit for developers is that the System Object Model is an emerging industry standard being implemented on other major operating systems. This simplifies cross-platform development, because SOM library source code is easily ported across operating systems. For example, the developer of a graphics program plug-in could package the code that performs the product's mathematical calculations as a SOM class library in Mac OS 8. The developer could then reuse the source code for this library when creating products that run on other SOM-supportive operating systems.

IDL compilers are quickly being supplanted by direct-to-SOM compilers. These compilers allow developers to create their own SOM object files without first using the IDL compiler and creating IDL files. Direct-to-SOM compilers are likely to be available to Mac OS 8 developers soon.

EXTENDING SOFTWARE WITH SERVER PROGRAMS

Developers can also extend Mac OS 8 by creating server programs. For example, a developer can create a server program that watches for e-mail. Clients of this program could provide users with tools to read and reply to e-mail.

Similarly, developers can make their products more extensible by separating them into cooperative programs that interact with users and server programs that process data and perform time-consuming I/O operations. Then, a developer who wants to enhance a product's user interface capabilities could release a new version of the cooperative program, leaving the server program untouched. Likewise, the developer could extend the server portion of the product and leave the user interface portion alone.

Some System 7 developers have created system extensions that use 'INIT' files to poll the system for particular events and system states. When these conditions occur, the 'INIT' files respond accordingly. For example, an 'INIT' file might be used in System 7 to compress files as soon as they are a certain number of days old. Mac OS 8, as you'll read in Chapter 13, supplies a Trigger Manager that efficiently tracks these types of conditions and informs programs of their occurrence. Thus, the developer of this System 7 file compression 'INIT' can implement the product more reliably and efficiently in Mac OS 8 as a server program that uses the Trigger Manager.

Several Mac OS 8 services are implemented as server programs. For example, the Process Manager is a server program. To preemptively schedule the main tasks of all cooperative programs in a future version of the operating system, Apple could conceivably revise the Process Manager without requiring changes to Mac OS 8 cooperative programs or to other parts of the operating system.

EXTENDING SOFTWARE WITH THE PATCH MANAGER

In previous versions of the Mac OS, developers often modified the behavior of a particular system software manager by patching its routines in an A-trap table. In Mac OS 8, however, such patching is heavily constrained.

First, PowerPC CPUs don't use trap tables. Therefore, when software patches A-traps, Mac OS 8 must emulate an 68K CPU. The emulation necessary to support System 7 patching slows software performance considerably. Second, the Code Fragment Manager creates separate copies of per-process static data for most shared libraries supplied by Mac OS 8. Thus, it's very difficult for any software to gain access to system-wide state information with a patch. Furthermore, patches in Mac OS 8 work reliably only if their effect is local—that is, on a per-process basis.

For these reasons, Apple discourages developers from patching routines supplied by Mac OS 8. However, it's clear from experience that patching is sometimes necessary. Therefore, Mac OS 8 supplies a Patch Manager that defines a programming interface for patching any import library, allowing a developer to patch Mac OS 8 routines or routines in the developer's own product. When patching is necessary, the Patch Manager simplifies the job for developers, and the Patch Manager ensures that patches operate reliably.

Using the Patch Manager, developers can patch operating system routines for one process at a time. A developer can package a patch in a shared library, making it available to all programs on the system. The patch can then be instantiated in every process that calls the library. Even with the help of the Patch Manager, however, it's very difficult for a patch to gain access to system-wide state information.

COMPATIBILITY NOTES

Support for System 7 Patching

Mac OS 8 doesn't support the System 7 mechanism for patching routines system wide—that is, across all programs. However, Mac OS 8 does support the System 7 mechanism for patching routines on a per-process basis. Note that many system services have been revised in Mac OS 8, and a patch developed for System 7 isn't guaranteed to produce its intended results.

SUMMARY

Mac OS 8 provides several mechanisms for enhancing and updating software. In particular, Apple can easily update portions of Mac OS 8 by supplying users with revised versions of discrete shared libraries or server programs, while developers can modify various parts of Mac OS 8 by

- subclassing the SOM class libraries provided by the operating system
- creating server programs
- calling Patch Manager functions

Developers, in turn, can design extensibility into their products by

- supporting OpenDoc within their applications
- designing and packaging product features as OpenDoc part editors
- dividing their products into pieces that lend themselves to periodic revisions and packaging these pieces as import libraries or plug-ins
- designing programming interfaces that allow their applications to support plug-ins
- incorporating SOM class libraries into their products
- separating their products into cooperative programs and server programs
- using the Patch Manager to modify routines in existing versions of their products

OpenDoc allows users to mix and match part editors and thereby extend software capabilities according to their own needs and tastes. To allow users to extend their work environments with OpenDoc, developers can supply product features in part editors, and developers of large, standard applications can allow the embedding of OpenDoc parts within documents created by their applications.

The routines in a shared library are dynamically linked at execution time to the fragments that use them. Apple can thus revise operating system features packaged as shared libraries without requiring users to update the programs that use these features. By packaging various program features in separate shared libraries, developers can more easily enhance and update their own software products, too. For example, a developer can extend a digital-video editing program with plug-in versions of shared libraries that provide various editing effects.

Sometimes developers want to extend features supplied by Mac OS 8. In the past, for example, some developers have found it useful to modify the controls, windows, or menus provided by the Mac OS. To help these developers, Apple has used object-oriented designs for many of its services. The class libraries supporting these designs are based on the System Object Model (SOM). Developers who use object-oriented extensibility for their own products can use SOM as well.

Because the SOM classes in Mac OS 8 are implemented as shared libraries, Apple can use them to extend elements of the operating system without forcing users to reinstall it. Software products using those elements automatically inherit the new features without being recompiled.

Developers can also use server programs to extend and easily update their products and the entire operating system while taking advantage of efficient preemptive scheduling of server program tasks.

If developers need to modify individual routines defined in an import library, they can use the Patch Manager to make these modifications on a per-process basis.

PLANNING A PRODUCT FOR MAC OS 8

If you're a developer, you can take the following steps to take advantage of extensibility in your software products:

1. Add OpenDoc support to your existing product.
2. Reimplement your system extension (that is, INIT) as a shared library, a server program, or an OpenDoc part editor.

3. Plan how to separate product capabilities into shared libraries, server programs, or OpenDoc part editors to simplify the job of extending and enhancing your product.

10

Files and File System Navigation

The Mac OS 8 file system manages the organization, reading, and writing of data located on hard disks, CD-ROMs, removable media drives, and other storage devices. These devices can be available on a network or directly connected to the user's computer. For developers, the operating system offers several convenient programming interfaces for using the file system, including File Manager routines, the I/O functions of the ANSI C-library file, and—for compatibility with System 7 applications—the System 7 File Manager routines. Developers with unique needs can also use a programming interface to the Low-Level File System Services.

The Navigation Services introduced with Mac OS 8 provide a standard way for users to locate and manage data on storage devices. When a user saves or opens files, these services guide the user through the file system hierarchy of volumes, folders, and files. These services also help the user learn about the data contained in files. For example, the user might preview a small representation of the contents of a document before opening it, determine its file format, and see what comments might be saved with it.

Because the file system is a reentrant service, it takes advantage of the operating system's efficient multitasking capabilities. The CPU doesn't waste valuable cycles waiting for a file system request to finish.

KEY TERMS AND CONCEPTS

- The **file system** is the part of the operating system that manages the reading and writing of information located on all storage devices available to the user's computer system. At the most abstract programming level, the file system offers several types of programming interfaces allowing applications to read, write, and otherwise manage information stored on these devices. The file system organizes information hierarchically into volumes, folders, and files.
- A **volume** is a portion of a storage device formatted to contain folders and files. A hard disk, for example, may be divided into several volumes.
- A **folder** is a subdivision of a volume. A folder can contain files and other folders. Folders are also known as **directories.**
- A **file** is any collection of related information stored as a single entity on a volume. On a volume, a file is the smallest entity discernible by the user. The volume format determines precisely how that information is organized on a storage device.
- A **volume format** is the structure of file and folder information on a disk. Mac OS 8 supports several volume formats, including the hierarchical file system (HFS) and other industry standard formats, such as the file allocation table (FAT) file system used by DOS and Windows.
- A **volume format plug-in** is a shared library that organizes information on a storage device. To support different storage devices, a system may include several volume format plug-ins. The file system dispatches and routes information between volume format plug-ins and programs that manipulate files and folders.
- **HFS** (hierarchical file system) is Apple Computer's standard volume format. HFS organizes files and folders in a hierarchical—that is, tree-like—structure.
- The **File Manager** defines the programming interface that most Mac OS 8 developers use to organize, read, and write data stored on volumes. Because the File Manager is reentrant, any task can call its programming interface.
- The **System 7 File Manager** is a cooperative service provided for backward compatibility with System 7 applications. Its programming interface can be called only by the main tasks of cooperative programs. (System 7 documentation from Apple Computer calls the System 7 File Manager simply the *File Manager.* In this book, however, *File Manager* refers to the newer, simpler programming interface that can be called by any task in Mac OS 8.)
- The **Low-Level File System Services** consist of a shared library that manages volume format plug-ins and provides a programming interface for application access to the storage devices connected to the user's sys-

tem. The Low-Level File System Services define a complex but powerful programming interface from which the File Manager, the System 7 File Manager, and standard C file–I/O routines are abstracted. Developers can directly use the Low-Level File System Services programming interface to build custom facilities for performing file I/O operations.

- Applications use the **Navigation Services** to present a standard human interface for opening and saving files.
- In Mac OS 8, a **panel** is a human interface object, such as a group of controls, a string of text, or a scrolling list, that can be placed inside a window. Inside a browser window for navigating the file system, the Navigation Services use panels to display document-related information.
- The **Alias Manager** helps developers create and use data structures for establishing and resolving permanent references to files, folders, and volumes.
- Programs can use the **Folder Manager** to determine or define the location of specially used folders. An example is the Fonts folder, where the operating system stores fonts for the user.

MAJOR POINTS OF INTEREST

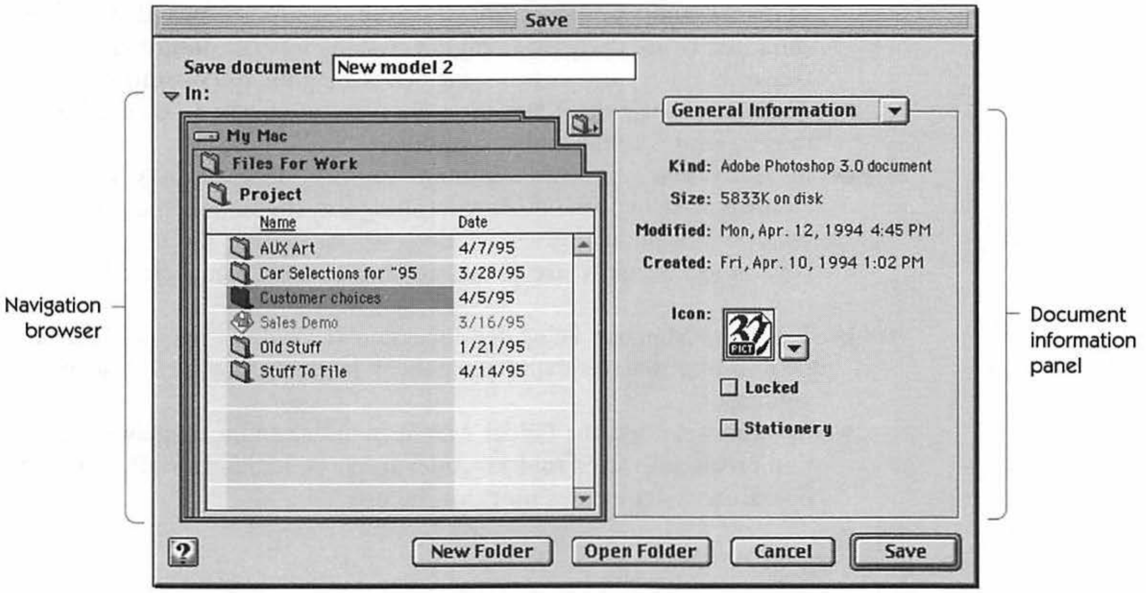
The Mac OS 8 file system is the portion of the I/O system that manages the reading and writing of information on storage devices. The file system organizes information into a hierarchical structure consisting of volumes (which may contain folders and files), folders (which may contain other folders and files), and files. The structure of this information on any particular storage device is determined by the volume format of that device.

Dialog boxes are windows used for special or limited purposes, such as soliciting information from the user before an application carries out the user's command.

An application uses a programming interface defined by the Navigation Services to manage the human interface for naming and identifying files. When invoked by an application, the Navigation Services display a dialog box that lets the user specify names and locations of files to be opened or saved. The Navigation Services provide a navigation browser by which the user moves between folders. The Navigation Services also display panels allowing the user to determine various kinds of information about files located with the navigation browser. These panels can contain such items as text, pictures, and controls. Figure 10.1 shows the navigation browser and the general information panel, one of several document-information panels that may appear in the Save dialog box.

When a user selects a file to open or chooses a name and location for saving a file, the Navigation Services report the user's choices to the application. To retrieve or store a file, an application then uses a programming interface defined by several services supported by the file system. These services include

FIGURE 10.1 Standard navigation browser and the general information panel



Name	Date
AUX Art	4/7/95
Car Selections for '95	3/28/95
Customer choices	4/5/95
Sales Demo	3/16/95
Old Stuff	1/21/95
Stuff To File	4/14/95

Kind: Adobe Photoshop 3.0 document
Size: 5833K on disk
Modified: Mon, Apr. 12, 1994 4:45 PM
Created: Fri, Apr. 10, 1994 1:02 PM

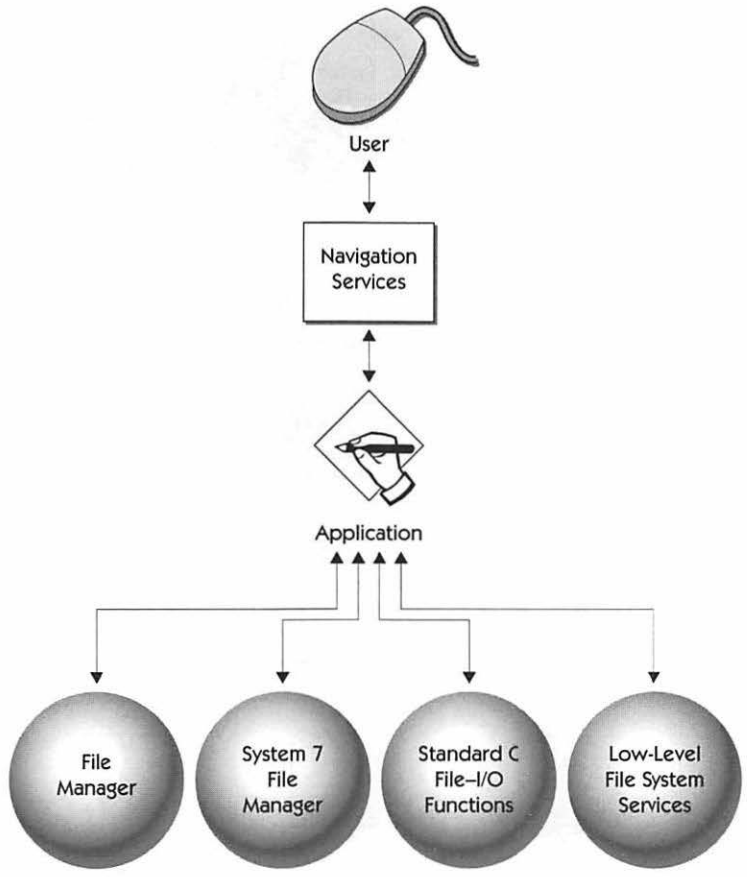
the File Manager, the System 7 File Manager, and standard C file–I/O functions. For example, an application can open a file using a routine defined by the File Manager or the standard I/O library for the C programming language. To provide additional programming flexibility, the file system also supports a Low-Level File System Services programming interface with which developers can build custom facilities for performing file I/O operations. This low-level interface is useful to developers of data-compression tools, disk-repair utilities, and other file system–intensive programs. Figure 10.2 illustrates the programming interfaces available to an application for managing files.

The File Manager, the System 7 File Manager, and the standard C file–I/O functions are all abstractions of the Low-Level File System Services programming interface. The shared library for the Low-Level File System Services, in turn, manages the volume format plug-ins for all storage devices connected to the user's system. The relationship between the Low-Level File System Services and the rest of the file system is illustrated in Figure 10.3.

Plug-ins, as explained in **Chapter 8,** are shared libraries dynamically prepared for use at execution time when the code fragments that need them call Code Fragment Manager routines.

The file system supports several industry-standard volume formats, and the extensible nature of the volume format plug-in architecture allows Apple to add support for other formats as they evolve. Depending on the particular storage device to which the program saves the file, the Low-Level File System Services invoke a volume format plug-in, which writes the data to the device. Figure 10.3 illustrates three possible types of storage devices: a directly con-

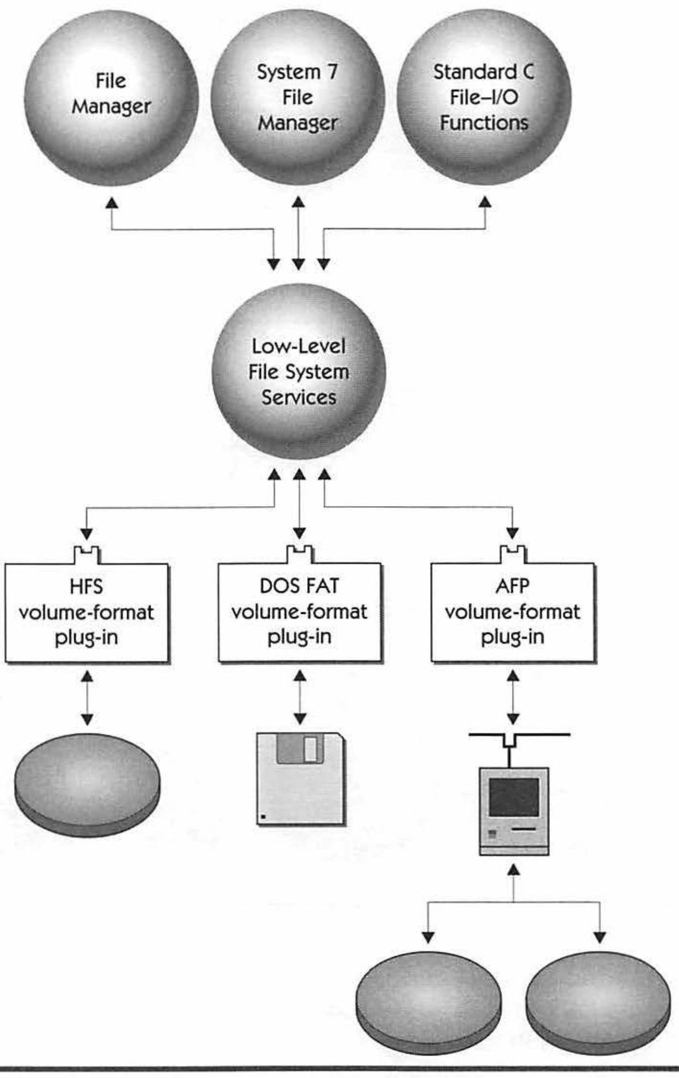
FIGURE 10.2 Application programming interfaces for file management



nected hard disk formatted as an HFS volume, a removable disk formatted as a DOS FAT disk, and a disk connected on a network using the Apple File Protocol (AFP).

Because it performs multiple operations concurrently, the file system makes efficient use of the Mac OS 8 multitasking capabilities. While a program waits for the file system to finish reading or writing a file, the microkernel schedules some other task to execute. The microkernel, as you read in Chapter 4, can also preempt one task accessing the file system and allow another task with a higher priority to execute. The newly executing task can access the file system even before the former task has finished its file I/O operations. All preemptively scheduled tasks are thereby granted concurrent access to the file system. As you read in Chapter 5, a developer can improve system efficiency by

FIGURE 10.3 Operating system components of the file system


File Manager
System 7 File Manager
Standard C File-I/O Functions
Low-Level File System Services
HFS volume-format plug-in
DOS FAT volume-format plug-in
AFP volume-format plug-in

threading an application so that one task, unrelated to the human interface, handles file access and other I/O-intensive operations.

When the main task of a cooperative program accesses the file system, it must generally finish reading or writing a file before another cooperative program main task can become eligible for execution. By comparison, when developers move file I/O operations from main tasks into additional tasks or into server programs, all cooperative programs can share system resources more efficiently.

Mac OS 8 supports the System 7 File Manager programming interface as a cooperative service even though the operating system supplies a newer, fully reentrant File Manager. Developers can use either programming interface to manage files, but the new File Manager provides much better system performance.

The virtual memory system, as you saw in Chapter 6, also uses the file system to efficiently supplement physical memory, allowing the user to open many large applications at one time.

MAC OS HERITAGE

Data Forks and Resource Forks

A file stored on a volume formatted according to the Hierarchical File System consists of two parts: a data fork and a resource fork.

The **data fork** for a document file contains data entered by the user; the application creating a document file can store and interpret the data in the data fork in whatever way's appropriate for that application. For an application file compiled to run on PowerPC-based computers, the data fork contains the application's code.

The **resource fork** contains data stored in **resources.** The data in a resource is interpreted according to its resource type. This data usually corresponds to data created by the developer for use by the program, but it may also include data created by the user while the application is running.

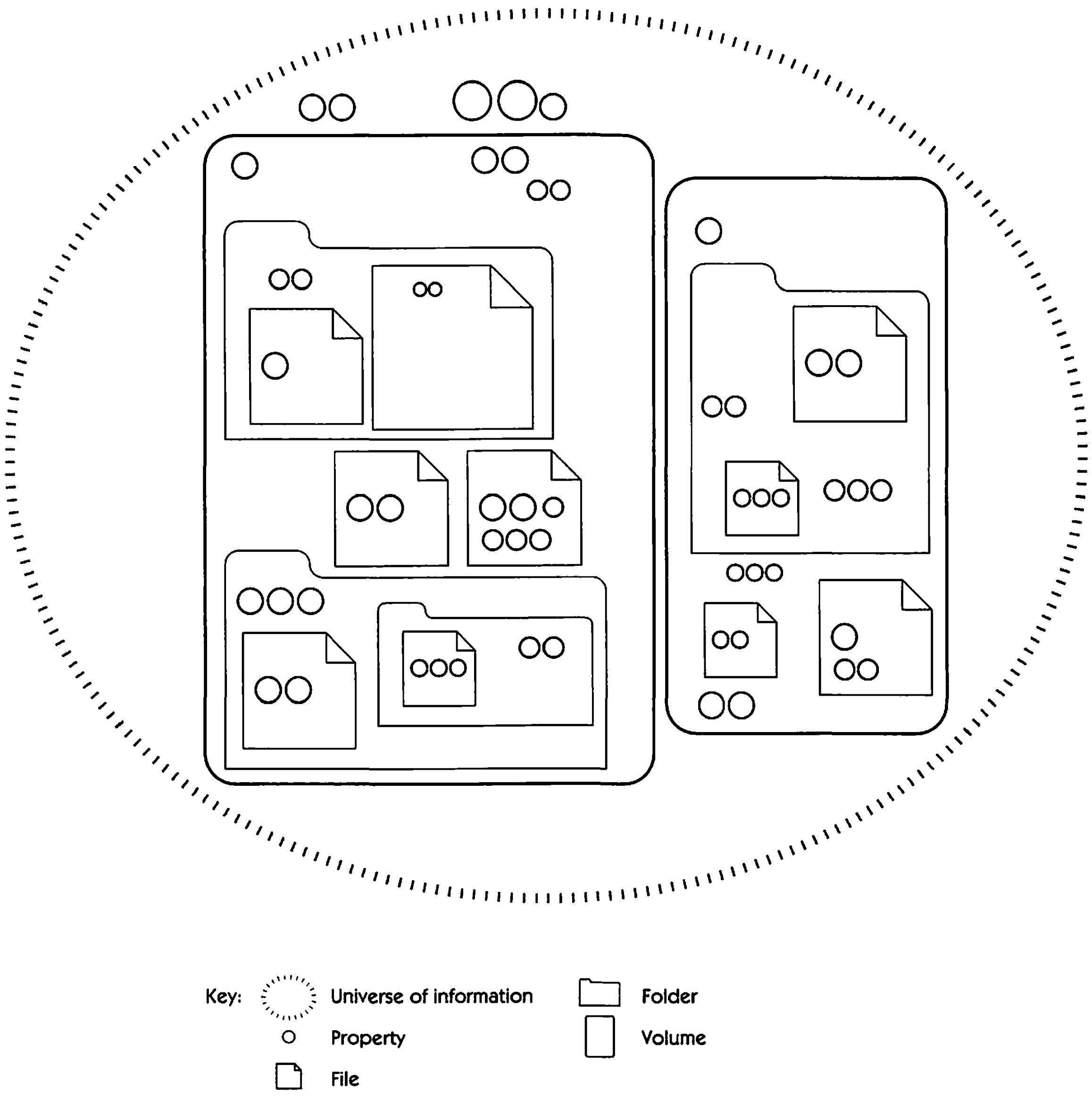
THE ORGANIZATION OF INFORMATION ON STORAGE DEVICES

The file system stores code and data so that programs can gain efficient access to this information. For efficiency, the file system uses a hierarchical model for information storage in which every element of the file system is potentially a container for information, as illustrated in Figure 10.4.

The most discrete element of the file system is called a property. All code and data is stored in properties. A **property** is a piece of information or a set of related information stored by the file system. Properties can be simple data items, such as dates, file types, and icon definitions; or they can be expandable sets of information, such as user-entered data. Each property takes up a certain amount of space allocated on a volume.

Related properties are grouped into an entity called a **file system object.** Files, folders, and volumes are examples of file system objects, and these may contain other file system objects as well as properties. For example, a folder

FIGURE 10.4 The file system properties and objects



can contain files, and a volume can contain folders and files. The Mac OS 8 File Manager defines data structures and constants that allow developers to determine and set such properties in a file system object as

- the object's creation date
- its modification date
- its name
- its size
- information about its forks

The file system defines a file simply as a collection of properties of any size or content. It is up to each specific volume format running under the file system to determine the meaning and content of each property. This generalized definition of a file allows the file system to support such diverse volume formats as HFS and DOS FAT.

THE PROGRAMMING INTERFACE TO THE FILE SYSTEM

Developers gain access to the file system with any of these related services:

- the File Manager for performing most file-related operations
- standard C functions for performing file I/O
- the System 7 File Manager, the programming interface supplied to support System 7 applications
- the Low-Level File System Services, allowing developers to create their own file access facilities for any special program needs

In addition, developers can use two services related to file system manipulation: the Folder Manager for determining or defining the location of folders—typically, special folders used by the operating system—and the Alias Manager for creating and using data structures that establish and resolve permanent references to files, folders, and volumes.

The File Manager

The File Manager defines the programming interface that nearly all Mac OS 8 programs use to access the file system. Because the File Manager is a reentrant service, any kind of task can use it. The File Manager can perform multiple file I/O operations concurrently, making the system perform as efficiently as possible. The operations that programs perform with the File Manager include

- creating, moving, and renaming files and folders
- opening and closing files

- getting and setting the properties of file system objects
- using stream or memory-mapped methods to access fork properties
- establishing a position in an opened file
- allocating and deallocating storage from a volume
- obtaining information about a specific file, folder, or volume
- iterating over file system objects to get information about them or to perform operations on them
- resolving path names and resolving object references for a given volume

To support developers creating products for international markets, the File Manager manipulates volume, folder, and filenames internally as text objects rather than as Pascal or C text strings. The Mac OS 8 File Manager can use text objects to handle file, folder, and volume names in any language. Described in Chapter 12, text objects can use any system of character codes, including Unicode. In addition to the character codes themselves, text objects incorporate information about the language system and the text-encoding system used for the names of file system objects. Because developers don't need to code the File Manager to use any particular language, they can adapt their applications for different language markets more easily.

Standard C File–I/O Functions

Programmers creating cross-platform products can conveniently gain access to the file system by calling standard C functions from the stdio.h library. The file system supports all file I/O functions defined for ANSI C. To extend ANSI C, the file system also supports additional file I/O functions defined by POSIX. These standard C programming language functions are also supported in System 7, but in Mac OS 8 they use the reentrant file system introduced with Mac OS 8.

ANSI is an acronym for the American National Standards Institute, an organization devoted to defining commercial standards, such as for programming languages like C. **POSIX** stands for Portable Operating System Interface, a set of standard operating-system services defined by the Institute of Electrical and Electronics Engineers (IEEE).

The System 7 File Manager

Mac OS 8 supports the programming interface defined by the System 7 File Manager. The routines in this programming interface are built on top of the Low-Level File System Services. Because access to the data structures maintained by the System 7 File Manager must be serialized, its programming interface can be called only from the main tasks of cooperative programs.

To be compatible with the Mac OS 8 file system, an application must use System 7 File Manager routines (and their associated data structures) strictly as documented in *Inside Macintosh: Files*. (In particular, the application should not directly manipulate the data structures or low-memory global variables used by the System 7 File Manager.) However, programs using the System 7 File Manager might not have access to any volumes except HFS-formatted volumes. (The new File Manager, by comparison, gives programs access to any volumes mounted on the user's system.)

Compared to the System 7 File Manager, the new File Manager contains about half as many routines, providing a simpler yet more powerful programming interface. The new File Manager is also much faster because it's optimized for execution on the PowerPC CPU and because better algorithms improve its performance.

COMPATIBILITY NOTES

Internal Data Structures of the System 7 File Manager

Many low-level data structures internal to the System 7 File Manager are changed in its reimplementation on top of the Low-Level File System Services. System 7 programs that modify these internal data structures won't work in Mac OS 8. Developers with special needs to alter System 7 File Manager characteristics should consider using the Low-Level File System Services or the Patch Manager, described in Chapter 9.

The Low-Level File System Services

The File Manager, the standard C file–I/O functions, and the System 7 File Manager provide simplified abstractions of the more complex programming interface defined by the Low-Level File System Services. Developers can also use this low-level programming interface to access the file system in their own products. Although the low-level programming interface gives more precise control over file system operations than, for example, the File Manager, it generally requires more calls. Most developers don't need to use the Low-Level File System Services. However, developers of some types of products—for instance, file-backup programs, disk-repair utilities, and network file-management tools—can take advantage of its very precise control over file system operations.

Some seldom-used file system operations are available only through the Low-Level File System Services. For example, only the Low-Level File System Services allow programs to mount and unmount volumes and to perform file operations on groups of files with a single call. Also, the Low-Level File System Services allow programs to make asynchronous as well as synchronous file I/O calls, whereas the File Manager routines are strictly synchronous. The multitasking capabilities of Mac OS 8 provide exceptional efficiency for preemptively scheduled tasks that perform synchronous file I/O, and the File Manager allows developers to make synchronous calls with minimal programming effort. But if a developer has a pressing need to perform asynchronous file I/O (for instance, to make a single asynchronous call from the main task of a cooperative program), the developer can still use the Low-Level File System Services.

The Folder Manager

Developers use the programming interface defined by the Folder Manager for determining and defining the location of folders—typically, those within the System Folder, which contains files and folders tracked by the operating system. (Because it doesn't perform file I/O, the Folder Manager isn't strictly part of the file system.) The Folder Manager allows programs to specify routing rules to be followed by the Finder and other client programs. For instance, a rule might specify that whenever the user drags a font to the System Folder, the Finder should place the font in the special Fonts folder. A program that needs access to the Fonts folder can also use the Folder Manager to quickly determine that folder's location. Developers who need to maintain their own folders in specific locations can also extend the Folder Manager for their own purposes.

The Alias Manager

The Alias Manager, available since System 7, establishes and resolves data structures, called **alias records**, that describe files, folders, and volumes. Aliases help users organize files for easier access. When the user chooses the Make Alias command from the File menu in the Finder, for instance, the Alias Manager creates an alias record to identify a file system object that the user might need to locate again. The Alias Manager has algorithms for using alias records to find files that have been moved, renamed, copied, or restored from backup. (Like the Folder Manager, the Alias Manager doesn't perform file I/O and therefore isn't strictly part of the file system.)

Programs, too, can create aliases. For example, a spelling checker application might create an alias record for a file containing a user-customized dictionary and store this alias with a word-processing document. When the user later runs the spelling checker on the document, the Alias Manager finds the customized dictionary, even if the user has moved it.

VOLUME FORMATS AND VOLUME FORMAT PLUG-INS

The File Manager doesn't include any code, or any routines in its programming interface, specific to a particular volume format. Instead, the file system specifies a message protocol for volume-format plug-ins that handle the I/O for specific volume formats. The file system supplies volume format plug-ins supporting a variety of industry standard volume formats, including

- HFS
- AppleTalk Filing Protocol (AFP), which allows access to AppleShare and Personal FileShare volumes over a network
- DOS FAT, the volume format used by the DOS and Windows operating systems
- These common CD-ROM formats: High Sierra, ISO 9660, Photo CD, and Audio CD
- Other third-party formats, such as Novell's Netware

Apple Computer has made the volume format architecture extensible. If the need arises, Apple can provide additional volume format plug-ins in the future.

Most developers don't need to think about what volume formats might be available. Instead, developers simply use the programming interface to the file system, and the Low-Level File System Services dispatch file I/O operations to the volume format plug-ins supporting the user's storage devices. For developers of products requiring information about a volume's particular format, such as disk-repair utilities, the File Manager provides routines for determining properties specific to various volume formats.

To take advantage of contemporary storage technology, the file system supports volumes and forks up to 2^{63} bytes in size. Up to this limit, the sizes of the volumes and forks that the file system can support are limited only by the capabilities of a specific volume format. For example, HFS in Mac OS 8 supports 2^{48}-byte—that is, 2-terabyte (TB)—volumes and 2-gigabyte (GB) forks.

MAC OS HERITAGE

The HFS Volume Format

The HFS volume format initially became available with System version 3.2, which was introduced with the Macintosh Plus computer. The maximum supported size for volumes and forks in HFS has grown considerably since its introduction. By System 7.5.3, HFS reached its current maximum volume size of 2TB and maximum fork size of 2GB. By comparison, the maximum volume and fork sizes in System 7.1 are 2GB each. In System 7.5, the maximum volume size is 4GB, and the maximum fork size is 2GB.

THE FILE SYSTEM AND THE VIRTUAL MEMORY SYSTEM

The file system and the virtual memory system interact to make efficient use of physical memory and secondary storage. When the operating system launches an application, as you may recall from Chapter 6, the virtual memory system maps all files containing code needed by the application into logical memory. The virtual memory system then transfers portions of these files from disk into physical memory when the CPU needs them, and it purges from physical memory any code that the CPU doesn't need. The disk files of programs are thus used as backing store for code not immediately needed in physical memory. By memory-mapping executable files in this way, the virtual memory system supplements physical memory without allocating additional disk space. The virtual memory system also uses the file system to manage scratch files containing temporary data not associated with a permanent disk file. To minimize the amount of disk space needed for this type of backing store, the virtual memory system dynamically expands and shrinks scratch files according to the immediate needs of programs running on the system.

The operating system maintains a minimum amount of physical memory for use as disk cache. The operating system also uses any additional physical memory that may be available to dynamically supplement this disk cache. The file system and the virtual memory system share the disk cache. When multiple programs open the same data file, whether by memory mapping it or streaming it, the file system gives them all access to the same data through this cache. Programs don't need to test how a data file has been opened, because memory-mapped and streamed data are accessed in the same way. (For this reason, writing code to share data between programs is far simpler than in System 7.)

Disk cache is a portion of physical memory set aside to temporarily store frequently used information that's permanently stored on disk. Because it's faster for the CPU to read information from physical memory than from a disk, disk cache helps programs run faster.

THE NAVIGATION SERVICES

The Navigation Services ease users through the task of navigating the universe of information saved on connected storage devices. Using the programming interface defined by the Navigation Services, applications can customize the system's browsing interface to give users access to document-specific information. When the user chooses a file to open or a location and name for saving a file, the Navigation Services report the choice to the application. The application then uses a file system routine to open or save the file specified by the user.

Programs can use both the File Manager and the System 7 File Manager in conjunction with the Navigation Services. The Navigation Services accept the data structures defined by either programming interface. Even unmodified System 7 applications automatically make use of the Navigation Services when soliciting user choices for opening and saving files.

To gain access to documents and document-specific information, a user typically opens the File menu for an application and chooses a command such as

A **document** is any piece of work that the user saves as a separate file. The user creates documents using cooperative programs.

- ▶ the New command to create a new document
- ▶ the Save command to save a new document
- ▶ the Open command to open a previously saved document

- the Save A Copy command to create a copy of an open document (this command is the same as the System 7 Save As command)
- the Find Documents command to open a Find window for performing relational searches of document names and contents across the entire file system

The search-and-save capabilities of Find windows are demonstrated in the CD-ROM version of Figure 1.12 on page 15.

When a user chooses New from an application's File menu, the application creates a new, untitled document and immediately saves it to disk. The user controls the default location for such newly created documents. When the user chooses Save from the File menu to save an untitled document, the application gives the user an opportunity to change the name and location of the saved document.

When the user chooses the Save or Save A Copy command, an application uses the Navigation Services to display the Save dialog box. When the user chooses the Open command, the application calls on the Navigation Services to display the Open dialog box. Both dialog boxes contain the navigation browser, from which the user chooses a file or file location. The navigation browser also makes it easy for the user to locate favorite items quickly, create new folders, and perform other common tasks. The navigation browser appears on the left side of either dialog box, and document information panels appear on the right. (See Figure 10.1 on page 146.)

The browser allows the user to navigate the hierarchy of nested folders. Developers can filter the types of files displayed in the browser and let the user choose which types of files to filter.

The general information panel, shown on the right side of Figure 10.1 on page 146, is similar to the Get Info window in System 7. This panel allows the user to manipulate selected document-specific information. (This panel also appears when a user chooses the Document Info command from the Edit menu in Mac OS 8.) The pop-up menu at the top of the panel allows the user to select which document-information panel to display in the dialog box.

The Open and Save dialog boxes offer a consistent human interface, allowing all applications to present the same navigation browser and system-supplied document-information panels. Both of these dialog boxes are extensible, and developers can supply custom panels. For instance, an application might supply a custom panel that displays the author, keywords, colors, or dimensions of a document selected in the browser. Developers can even embed OpenDoc parts into the panels displayed in the Open and Save dialog boxes.

COMPATIBILITY NOTES

Standard File Package

System 7 and earlier versions of the Macintosh Operating System offer the Standard File Package as a service by which developers present standard dialog boxes for saving and opening files. The Navigation Services provide a more consistent yet more versatile human interface for opening and saving documents than the System 7 dialog boxes do. In Mac OS 8, all program calls to the Standard File Package are fully supported so that they automatically display the Navigation Services browser in lieu of the older System 7 dialog boxes.

SUMMARY

The file system provides concurrent program access to storage devices such as hard disks and removable media. Any application can use the File Manager programming interface for opening, saving, and otherwise manipulating information on storage devices. For the benefit of programs that need additional control over file-system access, Mac OS 8 also supplies the Low-Level File System Services.

The System 7 File Manager programming interface is also provided as a compatibility service for System 7 applications. Only the main tasks of cooperative programs can use the programming interface of the System 7 File Manager, which performs more slowly than the new File Manager. To assist in cross-platform development, the Mac OS 8 file system also supports standard C file–I/O functions.

Applications use the Navigation Services to help users decide which files to open, where to save files, and what to name files. The Navigation Services supply all programs with a consistent browser, giving users an easy way to navigate the volumes and folders of all connected storage devices. These services also supply panels that display information about the files located with the navigation browser. These panels can contain text, pictures, controls, and OpenDoc parts, and developers can create their own panels for use with the navigation browser.

PLANNING A PRODUCT FOR MAC OS 8

If you are a developer, you can take the following steps to prepare products that take advantage of the Mac OS 8 file system:

1. If you've developed a System 7 application, make it native for PowerPC-based computers. Your application's file and other I/O operations will execute much faster in Mac OS 8.
2. Consider whether your application consumes very much time processing file I/O operations. If it does, separate your file I/O code from the

rest of your application. You can then implement this code more easily as a separate thread of execution in a multithreaded program.

3. Don't assume a maximum filename length in your code. For example, whereas HFS allows 31-character filenames, other volume formats impose different length limits.
4. Depending on the amount of memory you have and the speed of the device you are using, consider reading and writing file data in multiples of at least 16K. Reading an entire file into a buffer with one call requires less overhead than reading with multiple calls.
5. If you've already created a System 7 application, remove from your code any assumptions about the precise locations of any human interface elements in the Save and Open dialog boxes. You'll be able to use the relative position of the standard System 7 elements to determine the locations of Mac OS 8 elements.

11 Architecture of the I/O System

The I/O system is the portion of the operating system that transfers data to and from peripheral devices, such as hard disks, modems, speakers, keyboards, and pointing devices. Chapter 10 described the features of the file system portion of the I/O system. This chapter provides an overview of the entire I/O system.

In Mac OS 8, programs gain access to hardware only through a programming interface that insulates the I/O system from application-level software. This insulation protects hardware devices and the low-level code that controls them from errors in application-level software. To enhance system stability further, memory protections prohibit application-level software from changing and possibly corrupting the data used by the I/O system.

The use of a programming interface to insulate applications from the I/O system is one facet of the modular design of the I/O system. All portions of the I/O system are modularized by well-defined programming interfaces, making it easier for developers to extend and differentiate the platform for different hardware products.

All portions of the I/O system, including the device drivers controlling individual hardware devices, are compiled to run on the PowerPC CPU—there is no 68K CPU emulation to impede the performance of code that controls physical devices. Further optimizing system performance, the I/O system is completely concurrent, taking full advantage of the operating system's efficient multitasking capabilities.

KEY TERMS AND CONCEPTS

- ▶ An I/O **family** is a collection of software that provides a distinct I/O service to software clients. An I/O family typically consists of a privileged server program supported by a collection of shared libraries. The file system and the Open Transport networking services are examples of I/O families. Often, a family is associated with a set of devices with similar characteristics, such as storage devices or networking devices. Apple Computer supplies most families; developers of peripheral devices supply I/O plug-ins that integrate their devices within I/O families.
- ▶ As described in previous chapters, a **server program** has no direct interaction with users and, typically, provides services to other programs along the client/server model. A **privileged server program**, such as one used by an I/O family, differs from a nonprivileged server program in two major ways. First, a privileged server program runs when the CPU is in supervisor mode (giving the program greater access to computer resources), whereas a nonprivileged server program runs when the CPU is in user mode (protecting the system from errors in the program's code). Second, a privileged server program operates on data in a protected system-wide memory area reserved for use by privileged code, whereas a nonprivileged server program operates on data only within its own protected address space.
- ▶ An I/O **plug-in** is a plug-in (that is, a dynamically loaded shared library) that provides a particular implementation of the service offered by an I/O family. Within the file system I/O family, for example, a volume-format plug-in implements file system services for a specific volume format.
- ▶ A **device driver** is a type of I/O plug-in that directly controls a hardware device, such as a disk drive.

MAJOR POINTS OF INTEREST

Previous chapters have described the concurrent nature of the I/O system, particularly the interleaving of multiple I/O requests—such as file system operations and network transactions—so that the CPU doesn't waste valuable cycles waiting for any single I/O request to be completed. In conjunction with the multitasking capabilities of Mac OS 8, the I/O system efficiently transfers data to and from peripheral devices, increasing overall system performance. Multithreading a cooperative program to perform I/O-intensive operations outside of its main task increases the program's efficiency and user responsiveness. For example, a multimedia authoring program can read data from a CD-ROM disk in one thread of execution, write data to a hard disk in another thread of execution, and yet provide highly responsive user interaction in a third thread of execution. Even when data is slow in coming from the CD-ROM or going to the hard disk, the program can remain busy executing other operations on behalf of the user.

Because the I/O system is modular in design, modifications to one portion don't necessitate changes to other portions. The I/O system is divided into different I/O families tailored for particular types of hardware devices. As you'll soon see, two layers of programming interfaces within each family further modularize the I/O system.

An I/O family provides a distinct I/O service to other software clients. The file system, for example, provides applications with file I/O capabilities. An I/O plug-in is a particular implementation of the service offered by a family. For example, the HFS volume format plug-in allows the file system to use storage devices formatted with HFS.

I/O plug-ins are a superset of device drivers. In Mac OS 8, a device driver is a shared library that communicates with and controls a hardware device. For instance, a device driver may cause a head on a disk drive to read data from the disk. Although all device drivers are I/O plug-ins, not all I/O plug-ins are drivers. For example, the HFS volume format plug-in is not a device driver, because it doesn't control a hardware device. By contrast, a block storage plug-in writing to and reading from an HFS-formatted block-storage device is a device driver.

Block storage devices read or write blocks of bytes as a group. Disk drives, for instance, typically read and write blocks of 512 bytes or more.

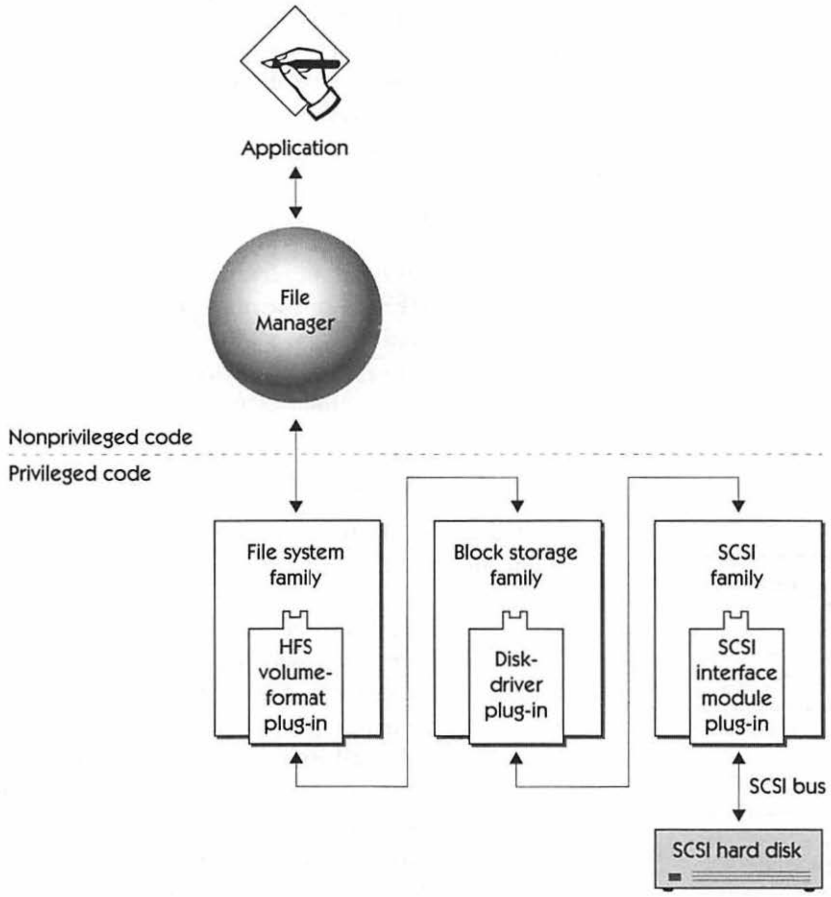
Figure 11.1 illustrates three I/O families that work together to complete a file I/O request from an application. An application uses the File Manager programming interface to read or write a file on a hard disk, which happens to be a SCSI device. The application request moves through the file system, which invokes a volume-format plug-in. The volume-format plug-in calls the block storage family, which in turn uses a disk-driver plug-in to call the SCSI family. A SCSI interface module plug-in moves the data over the SCSI bus connecting the storage device to the computer.

SCSI (Small Computer System Interface) is a data bus for connecting peripheral devices with computers. **PCI** (Peripheral Component Interconnect) is a bus architecture available on recent models of Mac OS–compatible PowerPC-based computers. **FireWire** is a new bus design nearing completion when this book went to press.

A client of an I/O family can be any code requesting the service offered by that family. A family's clients can include cooperative programs, other I/O families and their plug-ins, server programs, and the operating system. Figure 11.1 shows an application as a client of the file system family and the HFS volume format plug-in as a client of the block storage family. Each family provides two programming interfaces: one for its clients and one for its plug-ins. For example, the File Manager supplies a programming interface allowing client applications to gain access to the file system, but the file system defines a separate programming interface for its volume-format plug-ins.

Developers capitalize on this modular design to extend and differentiate the Mac OS 8 platform for different hardware products. Apple, for example, supplies I/O families for such diverse bus architectures as SCSI, PCI, and FireWire. Peripheral-device developers create plug-ins that integrate such products as video capture devices, scanners, graphics tablets, and laboratory equipment into these I/O families.

FIGURE 11.1 Interactions between an application saving a file and several I/O families



Supplying a modular, concurrent I/O system for Mac OS 8 required Apple Computer to implement an I/O system very different from that of System 7. This change has little impact on the compatibility of the vast majority of System 7 applications. System 7 applications that don't access hardware or hardware device drivers directly are compatible with the I/O system in Mac OS 8. However, with the exception of PCI drivers that closely follow Apple programming guidelines, System 7 device drivers that access hardware won't run in Mac OS 8. The section "System 7 Compatibility Issues" on page 172 describes how the new I/O system affects software products written for System 7.

MAC OS HERITAGE

The Macintosh I/O System

Before the introduction of System 7–compatible computers based on the PCI bus, the I/O architecture for Macintosh computers was based on resources of type 'DRVR' and their support routines. In this architecture, hardware developers write device drivers, usually in assembly language, and store their code in 'DRVR' resources. The Device Manager is the part of System 7 that controls the exchange of information between applications and device drivers. Applications generally gain access to devices through calls to the Device Manager or through calls to other managers, such as the File Manager, which in turn call the Device Manager. In System 7, applications can also directly manipulate devices. For example, some applications making extensive use of sound capabilities directly control the computer's sound chip instead of using the Sound Manager or the Device Manager.

I/O SYSTEM RELIABILITY FEATURES

To increase system reliability, Mac OS 8 protects the I/O system from application errors in three major ways:

- Memory-access permissions protect I/O system data from application-level software.
- I/O families are insulated from direct access by application-level software.
- All code in the I/O system resides (as does all code in Mac OS 8) in read-only memory areas where it's protected from possible corruption.

Privileged code is executed while the CPU is in supervisor mode. **Supervisor mode** is a state of operation for the PowerPC CPU that allows access to critical processor resources. **Nonprivileged code**, such as application-level software, executes while the CPU is in **user mode**—a state that protects certain critical resources, such as various processor registers, from being modified.

All I/O families and their constituent plug-ins are privileged, as indicated in Figure 11.1 on page 164. Because it's privileged, such code can deny application-level software any access to its data. This protection is available because, as you read in Chapter 3, Mac OS 8 allows a memory area to have one type of access permission for privileged code and another access permission for nonprivileged code. For example, a device driver might allow other privileged software, such as the microkernel, to write into a memory area where the device driver stores its data, but the driver might allow this data to be read-only for applications, thereby preventing applications from writing to and possibly corrupting this data.

Application-level software makes use of hardware only through the application programming interfaces defined by I/O families. For example, the application in Figure 11.1 can gain access to the file system only by using a client programming interface such as the File Manager. Hardware devices and their device drivers are thereby insulated from application-level software, protecting the I/O system from errors in application code. For example, a server program can continue using the I/O system to performing network operations even if another program using the I/O system were to crash.

Mac OS Heritage

Supervisor Mode in System 7

Mac OS 8 distinguishes between code that runs when the CPU is either in user mode or supervisor mode, but System 7 makes no such distinction unless the user turns virtual memory on. Even with virtual memory on, many supervisor mode instructions are emulated in System 7 for application-level software.

I/O Families

An I/O family is tailored to the needs of clients using a class of similar devices. For example, the Display Manager defines a programming interface tailored for display devices. When an application uses the Display Manager programming interface to change the number of colors displayed by video devices, the application indirectly invokes plug-ins that control those devices.

In its first release of Mac OS 8, Apple is providing the I/O families listed in Table 11.1, and potentially more. Peripheral-device developers can create I/O plug-ins that integrate their devices into these families.

Table 11.1 Several I/O families supplied by Apple Computer

file system family	SCSI family	block storage family
PCI-bus family	display family	real-time clock family
ATA-bus family	NuBus-bus family	sound family
Open Transport family	PC-card bus family	Device Manager family
Apple Desktop Bus family	NVRAM family (for physical memory)	user-input family (for keyboards, pointing devices, and other input devices)

Because the I/O system is modular and separated from the microkernel, computer manufacturers can add I/O families for newly developed peripheral-connectivity technologies. For example, Apple intends to supply an I/O family for the new FireWire bus architecture when it becomes finalized.

An I/O family typically consists of

- a shared library, called a **client library**, supplying a client programming interface
- a **family server**—that is, a privileged server program that receives, processes, and responds to service requests from clients of an I/O family and, usually, calls a plug-in to process these requests
- a set of plug-ins
- a **family expert**—that is, code that maintains information about the set of family-controllable services or devices and the family I/O plug-ins available on a given computer; this code can be part of the family server or be its own process

MAC OS HERITAGE

Programming Interfaces for Applications and Device Drivers

In System 7, there is only one kind of programming interface: the application programming interface (API). This makes all Mac OS services available to all varieties of Macintosh software. With the introduction of the PCI-based generation of Power Macintosh computers, System 7 began distinguishing between programming interfaces available to applications and those available to device drivers. In Mac OS 8, programming contexts have become increasingly specialized. For example, there are completely distinct programming interfaces for I/O clients and I/O plug-ins.

Client Programming Interfaces

Each I/O family provides a programming interface for its clients. This interface offers client access to services specific to that particular family. For the file system family, for example, the File Manager supplies a programming interface for client programs to access files; for the display family, the Display Manager supplies a programming interface for client programs to manipulate video devices. A shared library called the **client library** implements the client programming interface and forwards client requests for service to the family server.

A family may provide two versions of its client library, one for nonprivileged clients and one for privileged clients. For example, the nonprivileged version of the library may need to copy data between address spaces, but the privileged version, which has access to protected, system-wide areas of memory, may not. Both versions of a library present the same programming interface to all clients.

MAC OS HERITAGE

Direct Access to Device Drivers in System 7

Like Mac OS 8, System 7 provides a programming interface that insulates applications from hardware devices. Unlike Mac OS 8, however, System 7 doesn't restrict developers to using this programming interface. In System 7, application developers can access hardware devices directly in their programs.

Family Servers

A family server is a privileged server program that receives, processes, and responds to requests from an I/O family's clients. A client initiates a request by making a call to the client programming interface, and the client library forwards the request to the family server. The server then calls a plug-in to fulfill the request.

Plug-In Programming Interfaces

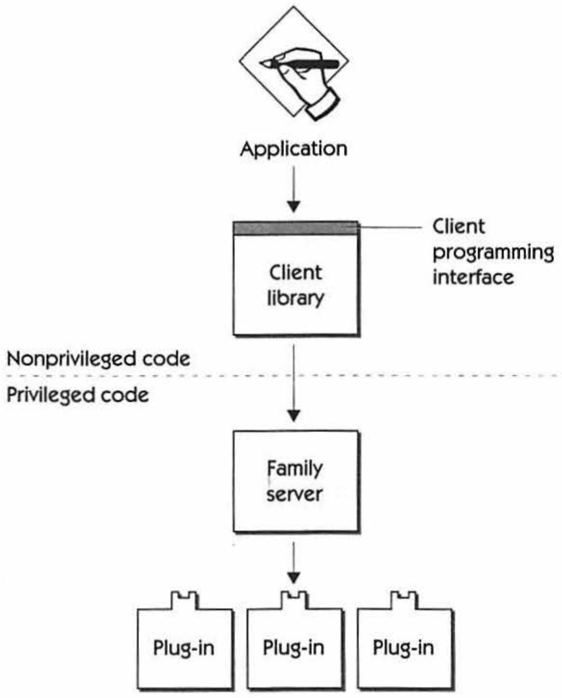
Whereas clients communicate with an I/O family through the family's client programming interface, plug-ins to an I/O family communicate with that family through a plug-in programming interface. Figure 11.2 illustrates the relationship of the constituents of an I/O family. In this example, an application uses the client programming interface of an I/O family—for example, the Display Manager. The client library then sends a message to the family server, which calls the plug-in to carry out the operation requested by the application, such as changing the number of colors displayed on a particular screen.

I/O Plug-Ins

Whereas most I/O families are supplied by Apple, developers of peripheral devices supply the I/O plug-ins that integrate their devices within I/O families. Device driver writers find that the specially designed set of services for a given family and its specially tailored programming interface simplify the creation of plug-ins. Developing plug-ins is further simplified because they can be written in a high-level language such as C.

A family server calls a plug-in in response to a request made by a client. An I/O plug-in usually consists of a main code section and a hardware interrupt

FIGURE 11.2 A client making a request for an I/O service



handler. The **main code section** contains the code that does most of the work of responding to client requests. All I/O plug-ins have a main code section.

A **hardware interrupt handler** is code registered with the operating system to service hardware interrupts. An I/O plug-in needs a hardware interrupt handler only if the plug-in responds to a physical device.

A **hardware interrupt** is an exception generated by a hardware device, notifying the CPU of a change of condition in the device. For example, a sound device may generate a hardware interrupt after fulfilling a sound output request. A hardware interrupt causes the microkernel to suspend the currently executing task while the CPU executes a hardware interrupt handler. The particular handler that's executed must have been previously installed by a plug-in to respond to the interrupts generated by that device. After an interrupt handler completes its operation, the microkernel resumes its preemptive scheduling of eligible tasks.

The plug-in programming interface specifies how interrupts are managed within a family. The I/O system architecture keeps the amount of code that can be executed at interrupt time to a minimum, thereby reducing **interrupt** latency, the interval between the generation of an interrupt and the execution of its interrupt handler. Minimizing interrupt latency improves overall system performance.

There are three main ways that the I/O system reduces interrupt latency:

- ▶ First, it prohibits application-level software from disabling interrupts, allowing interrupt handlers to respond to interrupts as quickly as they're generated. (System 7, by comparison, allows applications to disable interrupts.)
- ▶ Second, the I/O system limits the types of code that can run at interrupt time. Only privileged code can run at interrupt time. Applications and other types of nonprivileged code aren't allowed to run during a hardware interrupt. Whereas System 7 I/O completion routines, vertical blanking tasks, and Time Manager tasks run at either hardware interrupt level or as deferred tasks, Mac OS 8 runs these as nonprivileged tasks. (The historical box at the end of this section describes these System 7 mechanisms.)
- ▶ Third, and most importantly, the I/O system reduces the amount of work that a hardware interrupt handler performs. In response to an interrupt, a hardware interrupt handler performs only essential operations and defers as much work as possible to the plug-in's main code section.

Suppose, for instance, that a sound device generates an interrupt to signal that it's finished playing all of the sound data it's received and is ready for more data. The interrupt handler for that device might simply verify the interrupt, call the sound family with information about the device's status, and then clear the interrupt. The sound family could then use an I/O plug-in to send additional sound data to the device. As part of the task for the sound family server, the plug-in's main code section is preemptively scheduled for execution by the microkernel, keeping the system running as efficiently as possible.

MAC OS HERITAGE

Task Scheduling Outside of the System 7 Cooperative Multitasking Environment

Developers use various approaches to schedule their code for execution outside of System 7's cooperative multitasking environment. A common approach is to use a **completion routine**, which is executed as soon as an asynchronous call to some other routine is completed. Another approach is to use the Vertical Retrace Manager to schedule **vertical blanking tasks (VBLs)**, which can be executed any time a display screen is refreshed. A third approach is to use **Time Manager tasks** to schedule code execution independent of CPU clock speed or the occurrence of hardware interrupts. System 7 developers also use the

Deferred Task Manager to postpone the execution of lengthy operations at interrupt time by placing these operations in **deferred tasks**.

COMPATIBILITY NOTES

Vertical Retrace Manager, Time Manager, and Deferred Task Manager

Mac OS 8 provides compatibility support for System 7 applications that use the Vertical Retrace Manager, Time Manager, and Deferred Task Manager. However, Mac OS 8 doesn't support these services for device drivers. In lieu of these, the operating system provides **Timing Services** for scheduling the execution of device driver code at particular times.

Family Experts

Every I/O family includes code, called a **family expert**, that manages the information relating to the set of family-controllable services or devices and the set of associated plug-ins available on a given computer. A family expert uses various operating system services to collect and maintain this information; these services include

- the **Name Registry**, which stores information about hardware available on the user's system as well as the names, characteristics, and relations of various pieces of software on that system
- the **Driver and Family Matching Service**, which matches hardware-specific software with the I/O devices available on a computer
- the **Device Notification Service**, which alerts other parts of the I/O system of dynamic changes in device connectivity—for instance, the replacement of a laptop computer's disk drive card with a modem card

I/O families can be characterized as high level or low level. The characterization refers to the role the family expert plays at system startup when the connected hardware is being recognized and, after the system is up and running, when a device is added or removed.

A **high-level family** uses an expert that registers itself with the operating system to receive information about the availability of devices that can be controlled by the family. Based on this information, the family expert manages the availability of family plug-ins. The file system and Open Transport are examples of high-level families.

A **low-level family** uses an expert that has information about a specific piece of hardware, such as a bus or a main logic board. The expert knows how physical devices are connected to that piece of hardware and when a device is added to or removed from it. I/O families controlling buses—for instance, the PCI family and the NuBus family—are examples of low-level families.

The cooperation between the experts of low-level and high-level families permits Mac OS 8 to respond gracefully to changes in system configuration. As a result, the set of plug-ins known to and available through an I/O family always reflects the hardware currently available on a system.

SYSTEM 7 COMPATIBILITY ISSUES

Improved performance, reliability, and extensibility have been major design goals for Mac OS 8. To reach these goals, Apple Computer had to implement an I/O system very different from that of most System 7–compatible computers. The large majority of System 7 applications are compatible with the Mac OS 8 I/O system and benefit from its increased performance and reliability. System 7 device drivers written according to Apple guidelines for PCI-based Mac OS–compatible computers are also compatible with the new I/O system. However, all other System 7 drivers that access hardware will not run on Mac OS 8.

Application Compatibility

System 7 applications that don't access hardware or device drivers directly, but instead use programming interfaces for performing I/O operations, are insulated from the underlying changes in the I/O system. Mac OS 8 also provides compatibility support for System 7 applications that use the Vertical Retrace Manager, Time Manager, and Deferred Task Manager. (However, Mac OS 8 doesn't support these services for device drivers.)

System 7 developers need to revise their System 7 applications to run compatibly with the Mac OS 8 I/O system if they've ignored programming interfaces—such as those provided by the Device Manager, the Display Manager, the File Manager, and Open Transport—and have instead used nonstandard approaches to accessing hardware devices. In Mac OS 8, applications don't have direct access to hardware devices and device drivers. The only way for an application to use a hardware device or its driver is through an I/O family's client programming interface or through an interface maintained for compatibility.

As a compatibility service for System 7 applications, Mac OS 8 supports all of the functions described in the chapter "Device Manager" of *Inside Macintosh: Devices*. However, applications calling the Device Manager incur a performance penalty because they must go through a layer of compatibility code. For better performance and for access to services best suited to a given class of devices, System 7 developers should update their applications. Instead of using the Device Manager to access a device, developers should use the programming interface provided by the family to which the device belongs. Using the Display Manager, for example, an application benefits from using a set of routines tailored for display devices.

PCI Driver and Card Compatibility

A subset of the Mac OS 8 I/O system is implemented in versions of System 7 that support PCI devices on Mac OS–compatible computers. A System 7 device driver device written according to the guidelines offered in *Designing PCI Cards and Drivers for Power Macintosh Computers* is compatible with the PCI-based hardware platforms running Mac OS 8. (This book is available from Apple Computer at http://devcatalog.apple.com.) PCI cards with ROM-based versions of these device drivers are also compatible with PCI-based hardware platforms running Mac OS 8.

'DRVR' Compatibility

Mac OS 8 supports, through the Device Manager, emulated drivers of type 'DRVR' that do not touch hardware. An emulated driver, such as a print driver, is not an I/O plug-in. An emulated driver runs as nonprivileged code in the cooperative scheduling environment. System 7 device driver writers should note that in this new I/O architecture, a device driver is a PowerPC-native shared library that controls a physical device. Resources of type 'DRVR', protocol modules, application code, and system extensions of type 'CDEV' and 'INIT' are not part of the Mac OS 8 device driver model.

Device Manager Migration Path for System 7 Device Drivers

Mac OS 8 developers can create device drivers that make their services available through the Device Manager client programming interface. Such device drivers belong to the Device Manager family and are called **generic drivers.** The Device Manager family isn't tailored to the needs of any particular type of device.

Although the client programming interface defined by the Device Manager is more limiting than those defined by other families, the Device Manager family offers a migration path to driver developers who implement the basic changes required by Mac OS 8 without totally converting to the I/O architecture in Mac OS 8.

In addition to offering a migration path for System 7 device drivers, the Device Manager family allows developers to integrate a device into Mac OS 8 if no family exists for that type of device. Suppose, for example, that a PCI card receives, encrypts, and then returns data. An encryption family doesn't currently exist for Mac OS 8. By writing a generic driver for the card, a developer can incorporate the card into the Device Manager family.

SUMMARY

Optimized to take advantage of the PowerPC CPU's processing speed and capitalizing on the microkernel's multitasking capabilities, the I/O system quickly and efficiently transfers data to and from peripheral devices. Memory protection of I/O system data and insulation from application-level software through client programming interfaces make the I/O highly reliable. The modular design of the Mac OS 8 I/O system simplifies the work necessary for vendors to extend Mac OS 8 and differentiate their own value-added hardware systems.

The I/O architecture is built around families, which define I/O services suitably designed for similar types of devices. Programs request an I/O service by calling a client programming interface. The client library implementing that interface sends the request to a family server program, which processes the request and calls an I/O plug-in to carry out the request.

Device drivers are implemented as I/O plug-ins. A device driver that handles hardware interrupts spends very little time processing them at interrupt time. Instead, the driver performs only essential operations and defers all other operations to one or more preemptively scheduled tasks. Minimizing the amount of processing that occurs at interrupt time reduces interrupt latency and allows the system to run most efficiently.

Family experts track changes in system configuration so that the set of plug-ins known to and available through a family always reflects the hardware currently available on a system.

The I/O system is compatible with any System 7 application that doesn't access hardware devices or their device drivers directly. A System 7 application that directly manipulates hardware devices or their drivers must be modified to run in Mac OS 8. System 7 device drivers designed according to Apple guidelines for PCI devices are compatible with the Mac OS 8 I/O system. All other System 7 hardware device drivers must be rewritten for Mac OS 8.

PLANNING A PRODUCT FOR MAC OS 8

1. Consider whether your program consumes very much I/O processing time when it's not interacting with the user. If it does, separate your code into portions that perform user interface tasks and I/O operations. You can then implement these portions more easily as separate threads

of execution in a multithreaded program, allowing the operating system to perform concurrent I/O operations more efficiently.

2. If you've developed a System 7 application, make it native for PowerPC-based computers. This allows your application's file and other I/O operations to execute much faster in Mac OS 8.
3. Create device drivers for the PCI-based versions of System 7–compatible computers. If you follow the guidelines offered in *Designing PCI Cards and Drivers for Power Macintosh Computers*, these device drivers will be compatible with PCI-based versions of Mac OS 8–compatible computers. The work will also introduce you to the I/O architecture used in Mac OS 8.

12 Human Interface Toolbox

The Human Interface Toolbox underlies the Mac OS 8 user experience. The Toolbox is a collection of services that developers use to implement the standard portions of the Mac OS 8 human interface—for instance, windows, controls, and menus. Because programs use the Toolbox to display a consistent human interface, users can easily apply the skills they learn with one program to other programs.

The Human Interface Toolbox supports multiple sets of designs that coordinate the appearance of windows, menus, fonts, controls, and other onscreen objects. Choosing among these design sets, users can personalize the Mac OS 8 human interface to suit their moods and tastes. When several people share the same Mac OS 8–compatible computer, each user can set up a personal workspace characterized by its appearance, its level of difficulty, and other individual preferences.

All human interface objects are derived from a class library implemented with SOMobjects for Mac OS, allowing Apple and other developers to modify human interface objects while maintaining future compatibility between applications and the operating system. Although developers needn't use object-oriented programming techniques to implement standard Mac OS 8 human interface features, SOMobjects for Mac OS provides an object-oriented approach for developers who want to the customize and enhance these features.

For compatibility with older applications, Mac OS 8 fully supports the operating system services that implement the human interface in System 7.

System 7 applications that use standard menus, controls, and windows inherit the system-coordinated, user-selectable appearances of Mac OS 8. A System 7 application retains any custom human interface elements that it defines, but custom elements don't inherit the Mac OS 8 appearance.

KEY TERMS AND CONCEPTS

- ▶ The **human interface** comprises all of the facilities by which a user interacts with programs running on a computer. Because most human interface objects (such as windows, menus, and icons) are visual in the Mac OS, the term *human interface* is generally synonymous with *graphical user interface*. However, user voice input, sounds that alert the user, and other nonvisual elements are part of the human interface as well. The Human Interface Toolbox deals mostly with the graphical portions of the Mac OS 8 human interface.
- ▶ A **theme** comprises a coordinated set of human interface designs that determine the appearance of human interface objects on a system-wide basis.
- ▶ A **workspace** is one of several separate custom user environments for a single computer. A workspace is characterized by such user-specified attributes as theme, level of complexity, and application preferences.
- ▶ **SOMobjects for Mac OS** is the Apple Computer implementation of the System Object Model (SOM), an industry-standard architecture for the development and packaging of object-oriented software. SOMobjects for Mac OS provides the underlying technology for windows and panels, which are instantiated as objects derived from an easily extensible class library.
- ▶ A **class** is a generic structure used as a template for creating objects. Classes are defined in class libraries.
- ▶ An **object** is an execution-time structure that contains data and routines that operate on that data. An object is an instance of a class, which can be used to create additional instances that constitute separate objects.
- ▶ A **human interface object** is an object that encapsulates one or more human interface elements.
- ▶ A **window** is a human interface object that presents information such as a document or a message. A window usually contains other kinds of human interface objects called panels.
- ▶ A **panel** is any standard Mac OS 8 human interface object—for instance, a button, scroll bar, or editable text field—that can be placed in a window.

MAJOR POINTS OF INTEREST

A **control** is an onscreen object that the user can manipulate to take an immediate action or to change a setting to modify a future action.

Using the Human Interface Toolbox, developers create applications that present a consistent appearance and share predictable behaviors. For example, common controls such as pop-up buttons and progress indicators work and look the same in different applications. This human interface consistency helps users to learn new applications quickly and to be more comfortable—and therefore more productive—with a larger number of different applications.

As you've read in previous chapters, only the main tasks of cooperative programs can use the Human Interface Toolbox. When cooperative programs follow the Mac OS 8 event model (as described in Chapter 14), the Process Manager serializes their access to the Human Interface Toolbox. With all access to the Toolbox serialized, a program that begins a Toolbox-related operation, such as opening a new window, is allowed to finish that operation before another cooperative program can undertake a Toolbox-related operation. Thus, only one application at a time interacts with the user through the Human Interface Toolbox, keeping graphical interactions focused and predictable for the user.

The Human Interface Toolbox supplies applications with a variety of human interface objects, including

A **scroll bar** is a control that an application embeds in a window to allow a user to change the portion of a document displayed in that window.

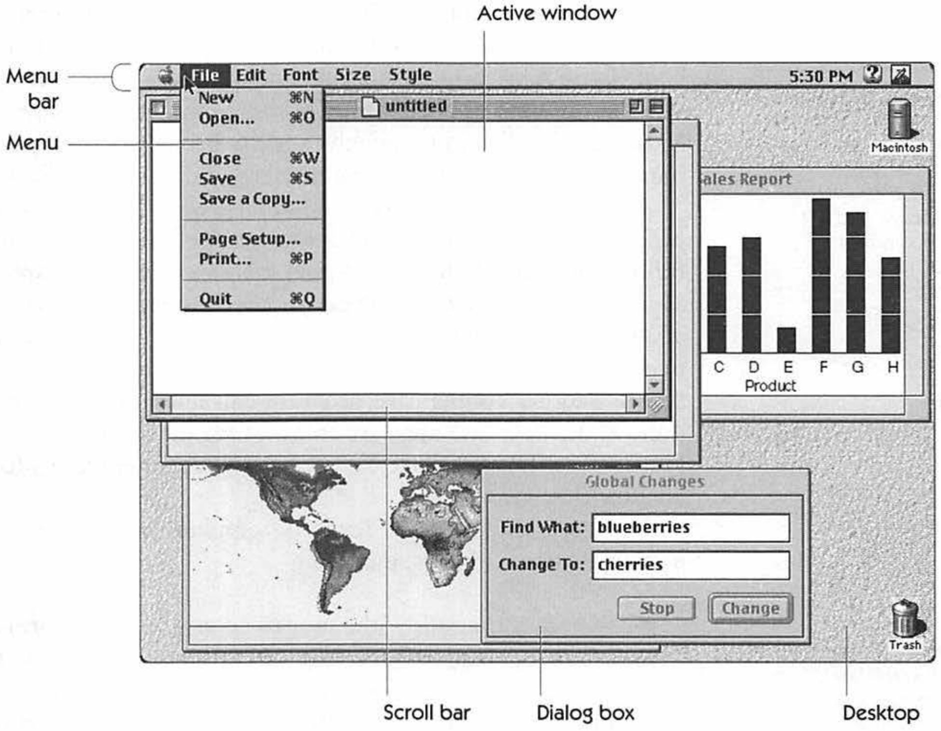
- windows that allow the user to enter and edit information
- controls, such as scroll bars and buttons, that allow the user to change application settings or invoke application actions
- dialog boxes that solicit information or decisions from the user
- menus allowing the user to choose commands

Figure 12.1 illustrates examples of all these objects as an application might present them.

All portions of the operating system that display a human interface use the Human Interface Toolbox, too. For example, the access and interview panels implemented with the Assistance Services (described in Chapter 13) and the navigation browser and information panels displayed by the Navigation Services (described in Chapter 10) are implemented with window and panel objects from the Human Interface Toolbox.

The Human Interface Toolbox supports customizing by each user in ways that maintain the overall look and feel of the human interface. For example, Mac OS 8 supplies multiple sets of design themes that affect the appearance and behavior of human interface objects. By switching themes, a user can change the appearance of all windows, controls, menus, and other objects displayed on a single computer. Figure 12.1 illustrates portions of the default theme, which is built into the Human Interface Toolbox of every system. Users can install or remove additional themes as they wish. Figure 12.2, for exam-

FIGURE 12.1 Typical human interface elements in the default theme

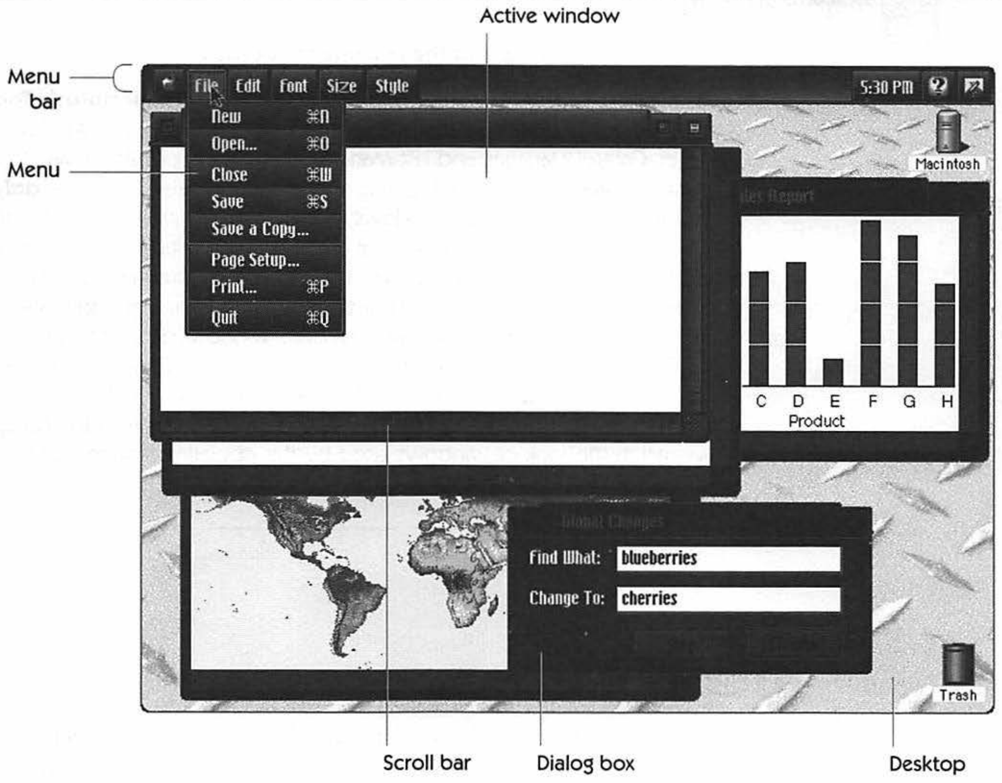


ple, shows human interface objects as they might appear in an alternate theme.

Mac OS 8 not only makes the human interface more flexible from a user's point of view but also makes it easier for developers to implement the human interface in their applications. For example, the Human Interface Toolbox simplifies the creation of software for the global market by allowing an application to display text within menus and window titles in any font and in any language.

To create an application's menus, windows, dialog boxes, and controls, a developer uses human interface objects. Implemented with SOMobjects for Mac OS, each human interface object knows how to draw itself appropriately depending on its state; for example, checkboxes mark and unmark themselves in response to user clicks in the boxes, sliders change their appearance in response to the user manipulating them, menus highlight correctly when the user selects them, and so on. Such built-in behavior simplifies software development efforts.

FIGURE 12.2 Typical human interface objects displayed in an alternate theme



If necessary to support specialized application needs, a developer can also reliably extend standard human interface objects or define new ones. As you may recall from Chapter 9, SOMobjects for Mac OS supports release-to-release binary compatibility. If a developer creates an enhancement to Mac OS 8 windows, for example, this enhancement will be compatible with future releases of Mac OS 8. Release-to-release binary compatibility also allows Apple to enhance Human Interface Toolbox features in subsequent releases of Mac OS 8, and these enhancements will be inherited automatically by all applications using the Toolbox.

Although the human interface objects are implemented using SOMobjects for Mac OS, it isn't necessary for developers to employ object-oriented programming techniques to display the Mac OS 8 human interface. Because of the language-neutral nature of SOMobjects for Mac OS, developers can write their programs in procedural languages, object-oriented languages, or mixtures of both and still use the Human Interface Toolbox.

COMPATIBILITY NOTES

System 7 Applications and the Macintosh Toolbox

System 7 developers use a group of libraries known as the **Macintosh Toolbox** to implement the Mac OS human interface. These libraries, including the Window Manager, Dialog Manager, Control Manager, and List Manager, are fully supported in Mac OS 8. Applications that use the standard System 7 definition procedures (also known as **defprocs**) for such human interface elements as windows, menus, and controls inherit the Mac OS 8 human interface appearance. Applications that use custom definition procedures work correctly on Mac OS 8. However, because custom definition procedures invoke their own drawing routines, Mac OS 8 can't draw these applications with the Mac OS 8 appearance.

For a System 7 application with its own window definition function, for example, Mac OS 8 displays that application's windows exactly as System 7 displays them. By comparison, a System 7 application using the standard window definition function inherits the appearance of Mac OS 8 windows. Even as the user switches themes, that application's windows match the look of windows used by the operating system and by other Mac OS 8 applications.

THEMES

As shown in Figure 12.1 and Figure 12.2, users can select different themes, which coordinate human interface designs across the entire system. Regardless of the theme, the core user experience remains the same. Users switch themes without having to learn new human interface metaphors. A theme determines the appearance of all human interface objects. Users can choose among the themes available to the system with a utility called the **Appearance control panel**, which also allows users to modify other aspects of appearance, such as the desktop pattern, highlight color, screen saver, and system font.

In addition to supporting user customization, themes insulate an application from future changes to the human interface. Because Mac OS 8 allows developers to deal with appearance abstractions rather than specific details, applications can support not only the new human interface designs in Mac OS 8 but also future design enhancements.

The **Appearance Manager** is the operating system service that provides the underlying support for themes. The Appearance Manager manages all aspects of themes and theme switching, including the Appearance control panel, support for a variety of color data, and support for animation and sound.

The Appearance Manager provides drawing routines that render the building blocks of human interface objects. If it's necessary to customize the look of any standard human interface objects, a developer can use Appearance Manager drawing routines to automatically coordinate the new appearance with user-selected themes.

In addition to these drawing routines, the Appearance Manager provides routines that allow an application to determine how the current theme draws various aspects of the human interface, such as the color of the menu bar. For example, an application can call the Appearance Manager to determine the current menu-bar color. The application can then coordinate the color of its window contents with that of the menu bar.

Users can also customize their systems by using **desktop animations.** A desktop animation can draw to a screen-saving window or to the desktop (that is, the area behind all windows, menus, and icons, as shown in Figure 12.1 on page 180). A desktop animation can be utilitarian (for instance, a ticking clock that periodically displays appointment reminders), informative (scrolling stock quotes or sports scores that have been downloaded from an online news service), or simply fun (a cartoon character performing pratfalls).

The Desktop Animation Manager cooperates with the Appearance Manager to establish and maintain user preferences for animations. When a user changes any desktop animation preference via the Appearance control panel, the preference takes effect immediately.

The user can install multiple desktop animations and select individual ones for different purposes—for example, one to use as a background when working at the computer and another to use as a screen saver when away from the computer. The Desktop Animation Manager allows a user with multiple monitors to select a different animation for each monitor or a single animation for all of them.

WORKSPACES

Because Mac OS–compatible computers are often shared by multiple users, Mac OS 8 allows users to set up several different workspaces for a single computer. A workspace maintains a user-customized environment that includes

- application and system preferences
- icons on the desktop
- desktop animations
- open Finder windows
- startup and shutdown items
- user's name and password
- the complexity of available menus and features; Chapter 1 describes the scalability of system complexity

If a system is configured for more than one workspace, the user is asked to choose a workspace either at startup or when switching workspaces. Each workspace is associated with a specific user name. If the workspace has a password, the user is prompted for it after choosing a name. A similar dialog box allows a user to switch workspaces after startup. Switching workspaces doesn't require restarting the computer, but it does cause all currently running applications to quit.

Mac OS 8 supplies developers with a Preferences Manager. Many applications allow users to set various preferences, such as the default font, pen width, menu content, file-backup behavior, and so on. The Preferences Manager gives developers a standard mechanism for controlling their applications' preferences. Using the Preferences Manager shields the details of preference management from developers and enhances their programs' capabilities in a multiple-workspace environment.

MAC OS HERITAGE

Single System Customizing in System 7

Although a single user can customize a single computer in many ways—including the arrangement of files and folders on the desktop, system preferences, and application preferences—System 7 can keep track of only one set of customizations.

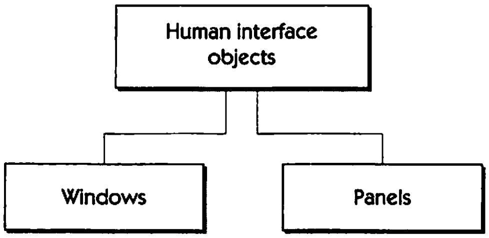
HUMAN INTERFACE OBJECTS

The Toolbox supplies developers with a class library defining a variety of standard windows, menus, lists, controls, editable text boxes, and other human interface objects. These objects automatically share the appearance of whatever theme the user selects. The class library for these objects is implemented as a shared library, using SOMobjects for the Mac OS.

To incorporate human interface objects into their applications, developers use their preferred programming languages to call functions exported by the Toolbox-supplied class library. Because of the language-neutral nature of the System Object Model, developers can use procedural as well as object-oriented languages to implement the Mac OS 8 human interface in their applications. A developer must use object-oriented programming techniques in order to modify any human interface objects. Since the Toolbox supplies so many types of objects, such modifications are unnecessary for most developers.

Even though it doesn't require a developer to write program code in an object-oriented programming language, the Toolbox-supplied class library

FIGURE 12.3 The top level of the inheritance hierarchy for the human interface objects class library



supports object-oriented techniques such as inheritance and subclassing. **Inheritance** is the transmission of properties and behaviors from one class to another. For example, all human interface objects inherit the same drawing methods. These methods cause all objects to be rendered onscreen in the currently selected theme. Closely related to inheritance is the technique of **subclassing**—the derivation of a new class from an existing class by adding to or overriding selected methods and data structures inherited from the original class. For example, a class for controls defines the behavior shared by all controls. Separate subclasses for scroll bars and push buttons add unique capabilities for each of these particular control types.

In object-oriented programming, a **method** is a function defined by a particular class.

In addition to inheritance and subclassing, the Toolbox-supplied class library also supports the object-oriented features of polymorphism and encapsulation. **Polymorphism** is the ability of client code to use the same method to call objects of different classes. For example, a program uses the same method to draw the text displayed by different classes of controls. **Encapsulation** is the packaging of an object's data and the routines that can act on it to protect the data from inappropriate changes. To gain access to an object's data, a client must call that object's programming interface.

A single superclass exists for all human interface objects. The methods defined in this superclass perform such operations as drawing the object, handling events involving that object, manipulating its location, and setting its visibility. Subclassed from this superclass are two more classes: one for windows and another for panels, as shown in Figure 12.3. These two subclasses inherit the data structures and methods of the superclass while defining methods and data structures of their own.

A hierarchical arrangement of additional subclasses further defines patterns of inheritance for the Human Interface Toolbox. For example, Figure 12.4 shows how additional subclasses are derived from the panels class. Additional subclasses are, in turn, derived from these. For example, subclasses of controls—such as scroll bars, push buttons, and sliders—inherit their common characteristics from the controls class.

FIGURE 12.4 Panel subclasses

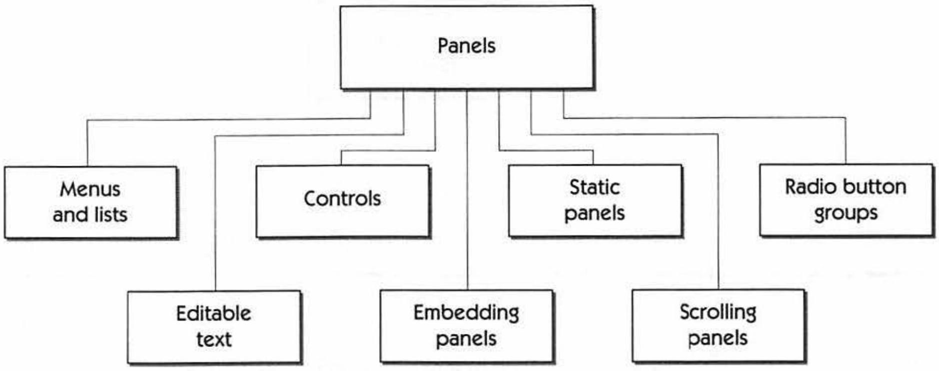
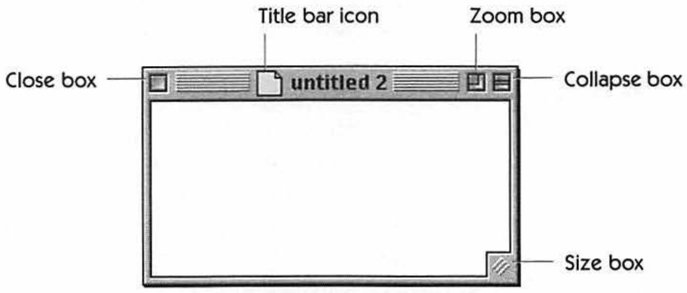


FIGURE 12.5 A standard document window



A particular execution-time instance of a class is an object. A window, for example, may contain two scroll bars, one vertical and one horizontal. Each scroll bar is an object derived from the same scroll bars class, and the window is an object derived from the windows class.

Windows

Windows are objects that programs use to present information, such as documents created by a user or messages directed to the user. When an application calls the Toolbox to create and display a window, the Toolbox instantiates an object from the windows class. The windows class includes methods that perform operations on windows—for instance, drawing, event handling, highlighting, ordering, positioning, sizing, and so on—automatically.

Figure 12.5 shows the elements of a standard document window object:

- A **close box** that, when clicked, dismisses the window.
- A **title bar icon** for use in drag-and-drop operations. For example, a user can drag a document's title bar icon to a folder on the desktop, then drop it to save the document in that location.
- A **collapse box** in the upper-right corner that the user can click once to hide all of the window (except the title bar) and then click again to redisplay the entire window.
- A **zoom box** next to the collapse box. When the user clicks the zoom box once, the window automatically expands to its optimal size on whichever screen is displaying most of the window. Clicking the zoom box a second time restores the window to its previous size and location.
- A **size box** in the lower-right corner that a user drags to resize the window.

To **drag** means to position the cursor on an interface element (such as a title bar icon), press the mouse button, and move the cursor to a new position. Dragging can have different effects, depending on what's under the cursor. These effects include selecting text, choosing a menu item, shrinking or expanding an object, or moving an icon.

To simplify localization, the Toolbox also supports resizing of windows in directions other than down and to the right, the use of multilingual text in window titles, and other features. (In Mac OS parlance, **localization** is the process of preparing a software product for a specific national or regional market.)

MAC OS HERITAGE

Application-Supplied Code for Handling Events in Windows

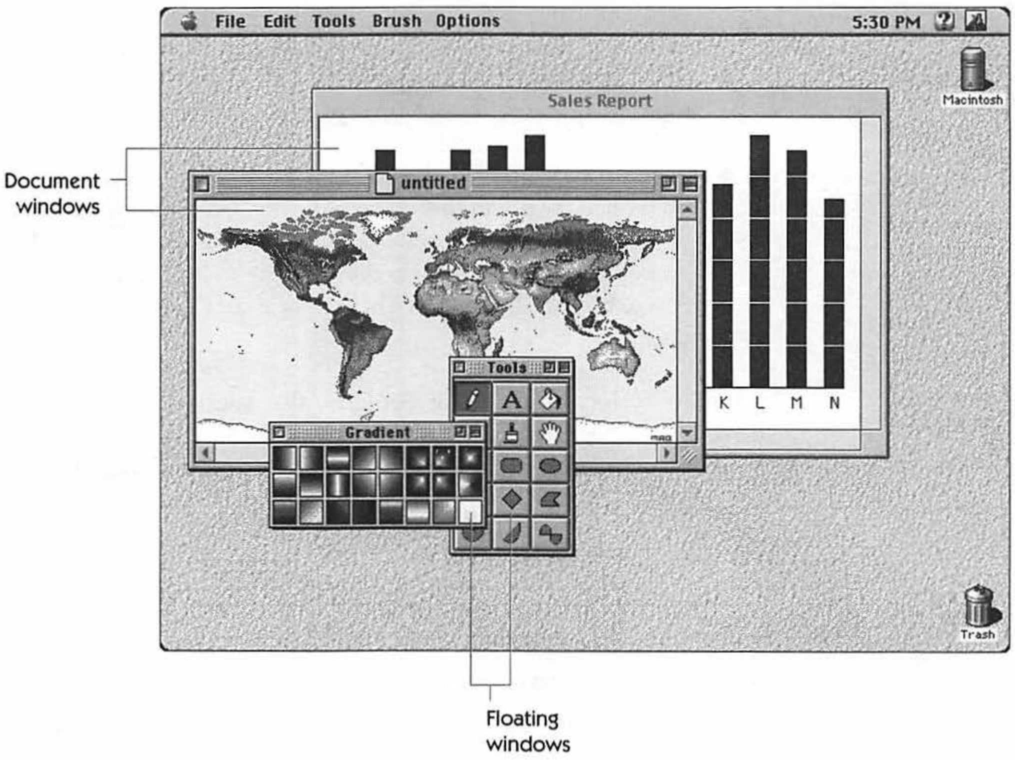
Much of the automatic behavior behind Mac OS 8 windows had to be programmed by System 7 developers. For example, when a user clicks the zoom box of a System 7 application, it must provide code that recognizes the event, determines which monitor should display the window, and then resizes the window accordingly. Mac OS 8 applications, by comparison, don't include this code, because windows created with the Human Interface Toolbox automatically resize themselves correctly in response to clicks in the zoom box.

Window Layers

Every application has a **layer** within which it displays its windows. Various system services, such as Apple Guide (described Chapter 13), control additional layers that may appear in front of or behind an application's layer. Developers can make use of these additional layers, too, when appropriate.

Every window in an application's layer belongs to one of three **sublayers**: a sublayer for modal windows, a sublayer for floating windows, or a sublayer for document windows. Each sublayer determines how a window appears in relation to other windows for that application.

FIGURE 12.6 Layering of floating windows and document windows



- ▶ **Modal windows** appear in front of all other kinds of windows in an application's layer. They're used for modal dialog boxes and alert boxes, both of which require immediate attention from the user. A modal dialog box or an alert box puts the user in the state or "mode" of being able to work only inside that modal window. The user can dismiss a modal window only by clicking its buttons.
- ▶ **Floating windows** appear in front of document windows and behind modal windows in an application's layer. They're used for tool palettes, catalogs, and other elements that let the user act on data in document windows.
- ▶ **Document windows** appear behind floating windows and modal windows in an application's layer. They're generally used for document data such as graphics and text and for modeless dialog boxes.

Windows always maintain this sublayering within a single application's layer. For example, the floating windows for the application shown in Figure 12.6 always appear in front of its document windows.

Window Uses

Various window types have specific uses. An application places the panel for an **alert box** inside a modal window to warn the user or to report an error. An alert box typically consists of text describing the situation and buttons that require the user to acknowledge or rectify the problem. The panel for a **modal dialog box** is also placed inside a modal window. A modal dialog box requires an immediate response from the user, such as providing information necessary for an application to carry out a command. By comparison, a user needn't respond immediately to the panel for a **modeless dialog box**, which an application places inside a document window. A user can move this type of window, make it inactive and active again, and close it. A **standard document window** is one in which the user enters text, draws graphics, or otherwise enters or manipulates data. An application may employ one or more **floating windows** that typically provide tools that let the user operate on data displayed in a standard document window.

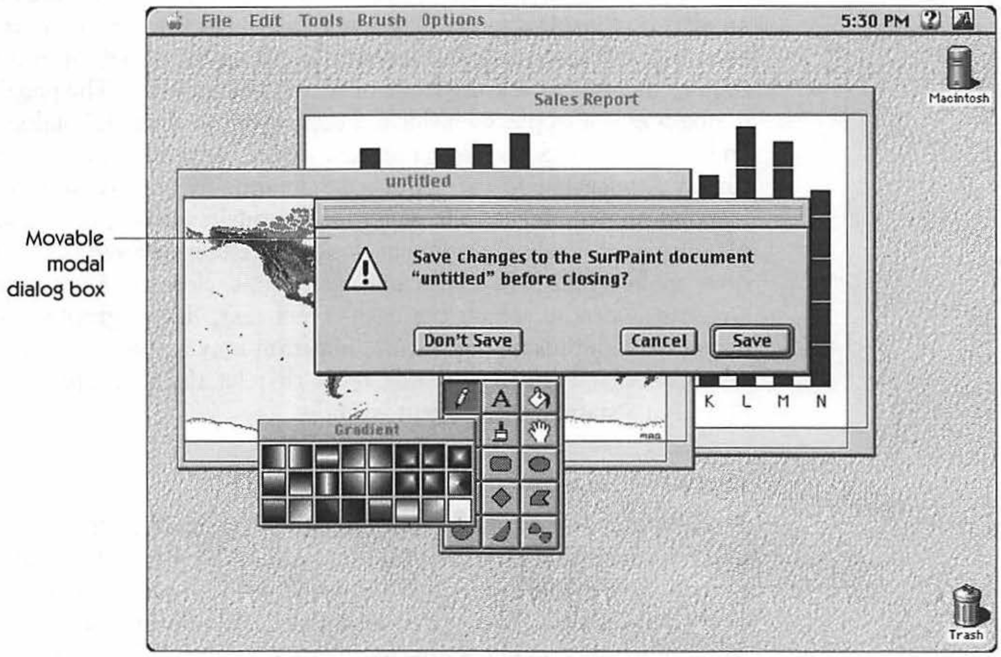
Window Activation

A user typically has one or more windows open, often from several different applications. However, only one window can be the active window. An application's **active window** is the frontmost modal or document window in the application's layer. (Floating windows can't be active windows.) It is the active window whose content is affected by user actions. The active window is identified by distinctive details that aren't visible for inactive windows; for example, the default theme displays title bars for active windows with characteristic "racing stripes." In Figure 12.6, the document window "untitled" is the active window. The other document window, titled "Sales Report," is inactive. If the user manipulates the floating windows, the corresponding actions affect contents of the active window.

When the user attempts to close the window "untitled" in Figure 12.6 without saving its contents, the application displays a movable modal dialog box, as shown in Figure 12.7. This modal window appears in front of all other windows in the application's layer, and it becomes the active window. Decisions made by the user with the aid of the dialog box apply to the document "untitled." The user can't manipulate the floating windows in this situation.

Windows automatically respond to any user action that can make them active or inactive. When the user clicks a window to make it active, for example, that window automatically activates its scroll bars without any special programming effort on the part of the developer. Similarly, an application's floating windows automatically disappear when the user switches to a different application.

FIGURE 12.7 A modal window in front of all other windows in an application layer



Window Groups

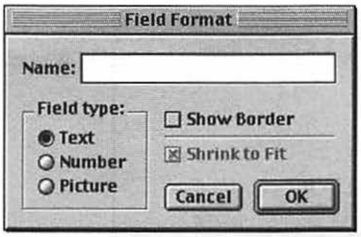
A **window group** is a useful abstraction that developers use to organize an application's windows and to automate many aspects of window management. A developer can add any application windows to a window group, regardless of the sublayers to which they belong. A developer can also add groups to groups.

A developer typically uses a window group to associate a window with one or more additional windows so that clicking the original window brings all of its associated windows to the front. For example, an application can use a window group to ensure that tool palettes associated with a document window also come forward whenever the user activates that window.

Panels

Panels are human interface objects that applications display in windows to help present and manipulate information. A panel manages all aspects of its own appearance and behavior as its state changes in response to user activities. For example, a scroll bar automatically redraws itself every time the user

FIGURE 12.8 A dialog box created from a window and multiple embedded panels



manipulates it. A developer supplies code that determines how changes in a panel's state affect an application—for instance, how to change the portion of a document displayed in a window in response to a click in a scroll bar.

Embedding panels and root panels play an important role in every application's human interface. As its name implies, an **embedding panel** contains other panels. Developers use embedding panels to assemble compound panels from the standard Toolbox panels.

A **root panel** is an embedding panel that fills a window's content area. The window passes all events that affect its content to the root panel. The root panel in turn passes events to other panels that it contains. For example, a modal dialog panel is a root panel that tracks user interaction with the panels it contains and takes care of all event handling required to enforce its modal state.

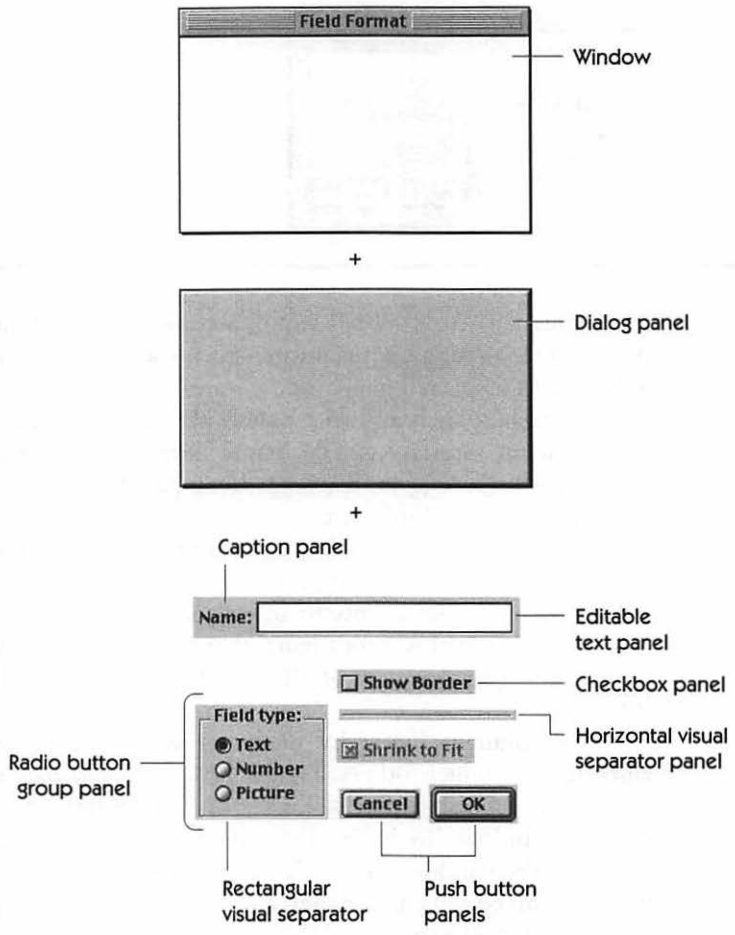
A **containment hierarchy** describes which human interface objects are embedded in which others. Figure 12.8 shows a dialog box created from a window and multiple panels. Figure 12.9 illustrates the containment hierarchy for this window. To begin, the window contains a dialog panel—the root panel for the window. Embedded in the dialog panel are several other panels, including an editable text panel, a radio button group panel, checkbox panels, and push button panels.

Dialog Boxes and Alert Boxes

All dialog boxes and alert boxes are created from specialized root panels, which an application displays inside windows that interact with the user. A dialog or alert panel controls all user interaction with its subpanels; for example, it tracks user input and maintains its own modal state, if any. A dialog or alert panel has a distinctive bevel just inside the frame of the window in which it is displayed.

Dialog Boxes An application displays a dialog box to solicit specific kinds of information from the user by means of the panels it contains, such as button panels, text panels, and list panels. Figure 12.10 shows an example of a **mode-**

FIGURE 12.9 Example of the containment hierarchy for a dialog box

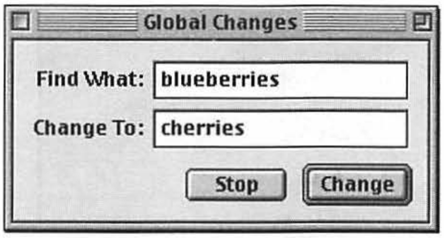


less dialog box. This type of dialog box uses a document window with no size box or scroll bars. The user can move a modeless dialog box, move between a modeless dialog box and other windows, and close a modeless dialog box.

By comparison, a **movable modal dialog box** consists of a dialog panel inside a modal window. It requires the user to work in a single mode within an application—that is, only inside the dialog box—until the user finishes interacting with that dialog box. Figure 12.11 shows an example.

A movable modal dialog box has a title bar that allows the user to move the dialog box around the screen—for example, to examine the part of the screen that it covers. The user can dismiss the dialog box only by clicking its buttons; however, the user can generally switch layers by clicking in another

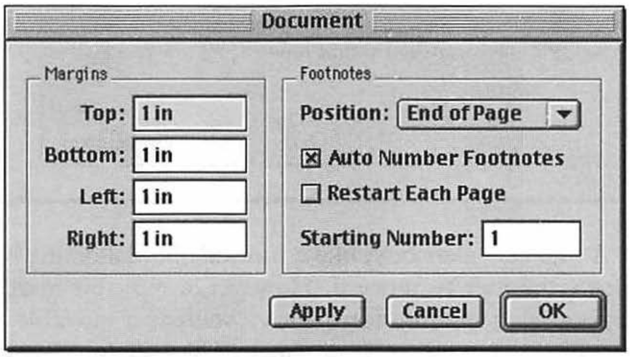
FIGURE 12.10 A modeless dialog box



Find What: blueberries

Change To: cherries

FIGURE 12.11 A movable modal dialog box



Top: 1 in

Bottom: 1 in

Left: 1 in

Right: 1 in

Position: End of Page

☒ Auto Number Footnotes

☐ Restart Each Page

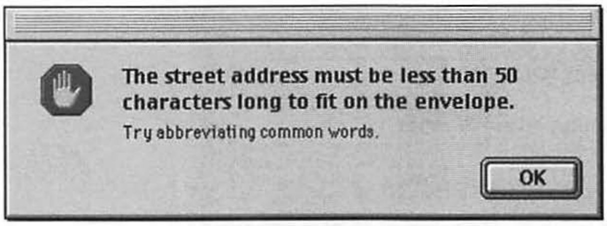
Starting Number: 1

application's window or by choosing another application from the Apple or Application menu. (A developer can prevent the user from switching layers if layer switching poses the risk of immediate damage to the user's data.)

For compatibility with System 7 applications, Mac OS 8 also supports nonmovable modal dialog boxes. A **nonmovable modal dialog box** resembles a movable modal dialog box except that it has no title bar. (Because a nonmovable modal dialog box restricts the user's ability to see portions of the screen, Mac OS 8 applications use movable modal dialog boxes instead.)

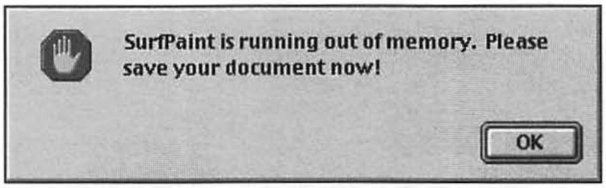
Alert Boxes An application displays an **alert box** to warn the user or to report an error. An alert box typically consists of an icon, text describing the situation, and buttons for the user to acknowledge or rectify the problem. An alert box contains an icon that helps communicate the seriousness of the information conveyed by its text. For example, the alert box in Figure 12.12 displays an icon indicating that the user must stop entering text because the address is too long to fit on an envelope.

FIGURE 12.12 A movable alert box



The street address must be less than 50 characters long to fit on the envelope.
Try abbreviating common words.

FIGURE 12.13 A nonmovable alert box



SurfPaint is running out of memory. Please save your document now!

A **movable alert box,** like a movable modal dialog box, has a title bar that allows the user to move it. However, a movable alert box can contain only text, an icon, and button panels, whereas a movable modal dialog box can contain any combination of panels. The user can dismiss a movable alert box only by clicking its buttons. As with a movable modal dialog box, a user can generally switch layers by clicking in another application's window or by choosing another application from the Apple or Application menu.

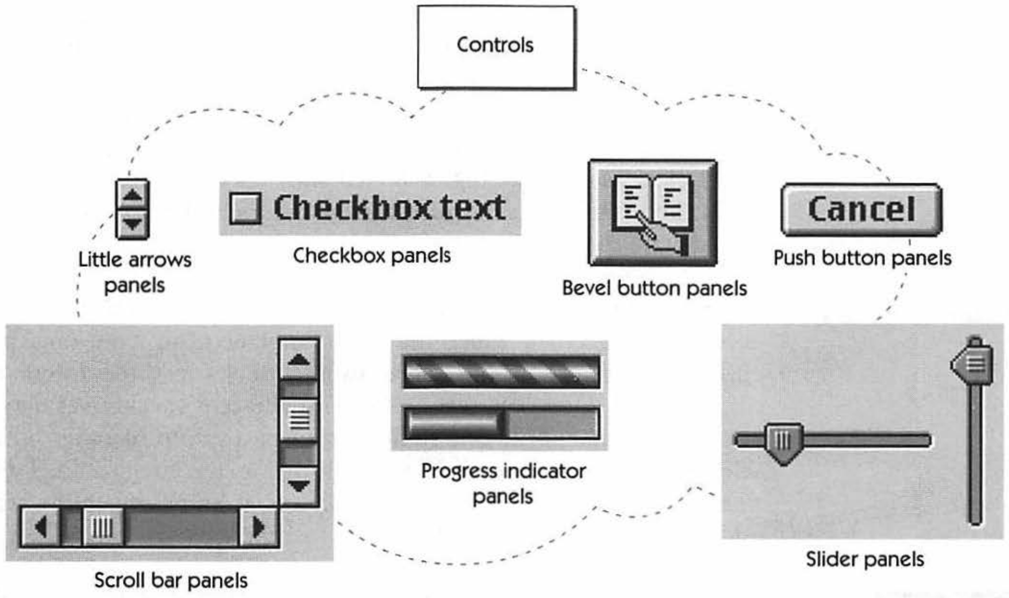
A **nonmovable alert box** resembles a movable alert box except that it has no title bar. As the example in Figure 12.13 shows, a modal alert box is displayed only when it's essential for the user to make an urgent decision or perform an immediate action.

Controls

Most windows—and certainly all dialog boxes and alert boxes—contain controls. Controls are panels that the user can manipulate to perform actions in an application or to change settings that modify future actions. Developers use a programming interface to get values from a control. For example, an application can get control values from a scroll bar in order to redraw the window's contents at the same time that a user is manipulating the scroll bar.

Figure 12.14 shows examples of panels created with standard subclasses of the controls class, which is itself a subclass of the panels class. The appearance

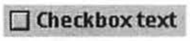
FIGURE 12.14 Examples of controls derived from subclasses of the controls class



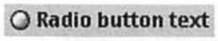
of each control is defined by its class and by the current theme. The examples in the figure and those in the margins are drawn in the default theme.

Push buttons A push button is a control that displays information (such as text, icons, or pictures) indicating its purpose. The Cancel button in the margin is an example. When the user clicks a push button, it performs an action instantaneously, such as canceling the operations defined by a dialog box. A developer can specify a push button as the default button, in which case it draws itself with the standard default appearance for the current theme (for example, with a ring around it). Then, when the user presses the Return or Enter key, the system responds as if the user had clicked the default button.

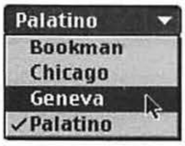
Bevel buttons Developers use bevel buttons for controls that, like push buttons, display information indicating their purpose. Developers commonly embed bevel buttons in tool bars and palettes. A bevel button can behave like a push button that lets the user press it once to perform an action instantaneously, or it can toggle between up and down—that is, unselected and selected—states.



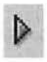
Checkboxes Developers use a checkbox to display a small square with a label consisting of text, an icon, a picture, or any other image. The label indicates what kind of setting the checkbox controls. A checkbox can display off, on, or mixed-state settings. The square is checked when the setting associated with the box is in effect, is empty when the setting is not in effect, and contains a short horizontal line when the setting is mixed. A mixed state indicates that a setting is in effect for some elements in a selection and not for others. For example, a checkbox that determines whether a character string is boldface appears in a mixed state if some characters in the string are bold and others aren't. The user can change a mixed-state checkbox to either the checked or unchecked state but can't directly change a checked or unchecked box to a mixed-state checkbox.



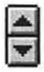
Radio buttons Developers use a radio button to display a circle with an accompanying label, which may consist of text, an icon, or a picture. Like checkboxes, radio buttons can draw themselves in three different states: on, off, or mixed. However, whereas several adjacent checkboxes may be selected at the same time, only one radio button in a group of radio buttons can be selected at any one time. In the default theme, the circle is filled when the setting associated with the button is in effect, is empty when the setting is not, and contains a short horizontal line when the setting is mixed.



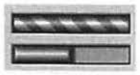
Pop-up buttons Developers use the pop-up button panel to display a menu associated with a button. When the user presses the mouse with the cursor over a pop-up button, additional menu items appear. Developers often use a pop-up button as an alternative to a radio button group or a list.



Disclosure triangles A disclosure triangle governs the display of items in a list, such as an outline containing subtopics. When the arrow of this control points right, only one item is visible beside it. When the arrow points down, both the original item and its subitems are visible in the list. To toggle between the two states, the user clicks the disclosure triangle.

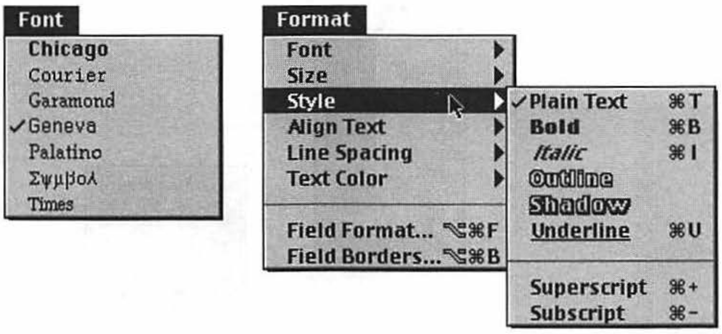


Little arrows This control displays a pair of arrows and typically accompanies a text box containing a numerical value, such as the date or time. Clicking the up arrow increases the value in the text box, and clicking the down arrow decreases it.

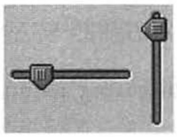


Progress indicators A progress indicator shows that a lengthy operation is taking place. The top example in the margin shows an **indeterminate progress indicator**. The revolving stripes on this indicator communicate that an operation is taking place, but this type of indicator doesn't show how long the operation might continue. To create a **determinate progress indicator**, such as the bottom example, a developer supplies values to the panel indicating how much of an operation has been completed. In response, the progress indicator

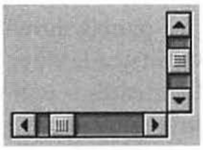
FIGURE 12.15 Two menus created from the standard menu panel class



fills itself from one end to the other, indicating what percentage of the operation has been completed.



Sliders This control displays a range of values, magnitudes, or positions. A movable indicator shows the current setting. Sliders, which can be vertical or horizontal, allow users to alter a value by moving the indicator up and down or back and forth.



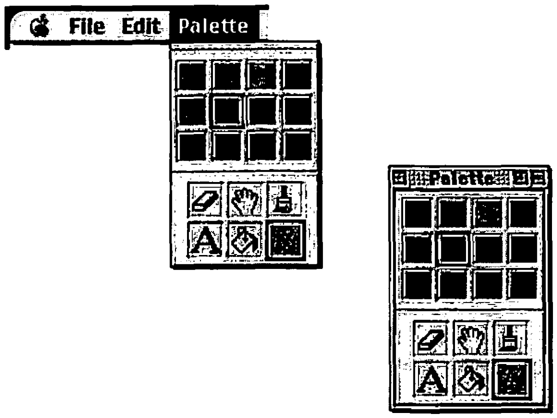
Scroll bars Windows, lists, editable text panels, and the like can have a horizontal scroll bar, a vertical scroll bar, or both. In the default theme, a scroll bar is a narrow rectangle with an arrow in a box at each end and a scroll box that moves between them. Users click the arrows or drag the scroll box to display more of the document by scrolling it into view. Generally, contents scroll at the same time that the user drags the scroll box.

Menus

A **menu** is a panel that lets the user view or choose an item from a list of choices or commands, as shown in Figure 12.15 and Figure 12.16. A developer designs an application's menus according to the tasks or actions that the application performs. Developers instantiate menus that can

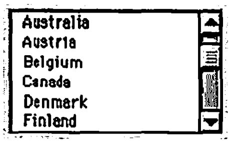
- contain images in columns and rows
- display menu items in any font in any language using any script
- contain submenus
- display keyboard equivalents for any menu item using multiple modifier keys
- can be torn away from the menu bar by the user and placed anywhere onscreen

FIGURE 12.16 A tear-off menu



▶ support a "sticky menu" mode that allows users to leave a menu or submenu open and choose menu items by clicking them or by typing on the keyboard
▶ respond automatically to user actions, such as by highlighting menu items when the user navigates through them

Like all panels, menus are displayed in windows. Unlike other panels, however, developers don't write code to instantiate these windows. Instead, when an application instantiates a menu, the Toolbox automatically creates a window for it, displays it in that window, and performs all window management on behalf of the menu.



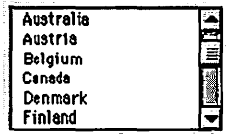
Lists

A list is a panel containing a series of items displayed in a rectangle. The example in the margin shows a list containing names of countries embedded in a scrolling panel (described in the next section). A list embedded in scrolling panel is called a **scrolling list**.

Each item in a list is in a rectangular cell. All cells in a list are the same size but may contain different types of data and multiple columns of data. The user clicks cells to select them. Developers can instantiate lists with cells that

▶ display text in any font and in any language
▶ contain icons, pictures, patterns, or other static images
▶ respond automatically to user actions, such as highlighting when the user navigates through them

A developer uses Toolbox-supplied functions to store and update the data in a list, display the list in a window, and respond appropriately to user interaction with a list.



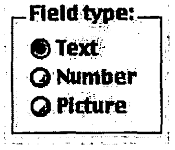
Scrolling Panels

A **scrolling panel** contains a vertical scroll bar, horizontal scroll bar, or both and is designed to contain another panel (such as an editable text panel or the list panel shown to the left) that's larger than the area allocated for the scrolling panel. Scrolling panels supply functions that allow a developer to set and get vertical and horizontal scroll values, vertical and horizontal scroll increments, and scroll bar visibility.

Editable Text Panels

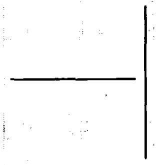
An **editable text panel** lets the user edit the text it displays. An editable text panel displays the contents of a text object (described later in this chapter) using the services of a **text engine**, a shared library that manages the formatting, drawing, and editing of the text in response to user actions and application calls to the panel. When instantiating an editable text panel, a developer specifies a text engine to use. Mac OS 8 provides a default text engine based on the TextEdit service familiar to System 7 developers, and a developer can provide or use other text engines according to the needs of the application.

The editable text panel itself is independent of the text engine associated with it. Although it can perform text-specific operations such as inserting, deleting, and replacing text, an editable text panel can also use its associated text engine to respond automatically to user input from the mouse, keyboard, voice-recognition software, and the like. An editable text panel also supports copy, paste, and drag-and-drop data sharing without any special programming effort on the part of developers.



Radio Button Groups

A **radio button group** encapsulates several radio button panels, such as those shown on the left. Unlike the individual radio button panels illustrated on page 196, a radio button group panel can handle mouse and keyboard interaction, including highlighting and the tracking of user interactions.



Visual Separators

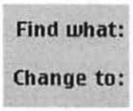
Visual separator panels display horizontal, vertical, or rectangular visual separators. The figure on the left shows examples of horizontal and visual separators. A rectangular visual separator can optionally include a title. For an example of a rectangular visual separator panel, see Figure 12.9 on page 192.

Static Image Panels

Embedding panels such as dialog boxes and palettes often include icons, pictures, patterns, and simple unstyled caption text. Although users don't interact with these elements, it is often convenient for developers to implement them as panels. (Alternatively, a developer can implement simple visual elements without the aid of the human interface objects class library by using imaging objects—described in the next section—or other lower-level operating services, such as the Icon Utilities or the QuickDraw graphics system familiar to System 7 developers.)

Icons An **icon** is a graphic representation of some human interface element, such as a document, disk, folder, or application. The icon in the margin shows the talking-face icon commonly used to identify alert boxes. A developer can display an icon as a panel. An icon panel encapsulates an image and draws itself within an embedding panel.

QuickDraw pictures A QuickDraw picture is an image described by a sequence of QuickDraw drawing commands that have been saved to a file. A developer can conveniently display a QuickDraw picture, such as the image of the dog in the margin, as a panel.



Find what:

Change to:

Caption text Static text that can't be changed by the user can be displayed in a static text panel, such as that shown in the margin. Developers use editable text panels (described on page 199) to create text panels that can be edited.

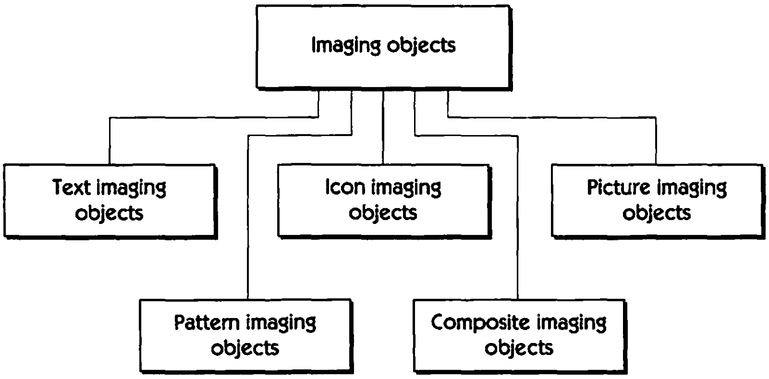
IMAGING OBJECTS

An **imaging object** is an object, derived from the imaging objects class, that can draw a specific kind of image data, such as text, icons, or QuickDraw pictures. The imaging objects class is an abstract superclass unrelated to the superclass for all human interface objects. The methods defined by the imaging objects class perform operations common to all imaging objects, such as creating, initializing, measuring, and drawing a new image. Derived from this class are separate imaging object classes for several commonly used types of image data. These subclasses are shown in Figure 12.17.

Human interface objects use imaging objects to draw

- ▶ the titles of windows, push buttons, checkboxes, icons, rectangular visual separators, and all other human interface objects that can have a title
- ▶ list items
- ▶ menu items

FIGURE 12.17 The inheritance hierarchy for the imaging objects class library



With an imaging object, a developer can also measure or draw an image without using the human interface objects class library at all.

Using the composite imaging objects class, a developer can bundle two or more images of potentially different types into a **composite imaging object**. For example, a composite imaging object could draw a file icon accompanied by a text filename.

PROGRAMMING CHARACTERISTICS OF THE TOOLBOX

All Human Interface Toolbox services support similar capabilities in similar ways, thus ensuring a consistent programming interface as well as a consistent user experience. The following sections describe the most important principles that underlie the Human Interface Toolbox from the developer's perspective.

Data Opacity

Developers don't have direct access to the underlying data structures of the Toolbox. Instead, developers use functions that manipulate these structures. The opacity of Toolbox data structures ensures that an application doesn't have to depend on Toolbox implementation details. In this way, Apple can refine the Toolbox in the future without requiring Mac OS 8 developers to rewrite their applications.

Data Extensibility

The Toolbox allows developers to add arbitrary data to Toolbox data structures without manipulating the structures directly. The Human Interface Toolbox supports this data extensibility through the Collection Manager, an operating system service introduced in System 7.5 with QuickDraw GX. Collection items can be used for a variety of purposes. For example, suppose a preferences dialog box allows the user to switch among several pages of preference settings, where each page displays multiple panels. A developer can use collection items to associate a page ID with the panels that appear in a particular page. This association makes it easy to hide or show the appropriate panels when the user switches pages.

Object Life Cycle Management

The Toolbox keeps track of multiple references to a single human interface object on an application's behalf, releasing operating system resources allocated to the original object only after all references to it have been released. Thus, an application doesn't need for the sake of system efficiency to track its own object references.

Integrated Support for International Text

At the time of this book's production, nearly half of Apple's revenues from Macintosh computers come from outside the United States. To help developers create applications for this international market and to help users collaborate with documents written in different languages, Mac OS 8 provides operating system support for international text management.

Designing Applications for International Markets

The process of designing and creating applications with various languages and cultures in mind is called **internationalization.** Building an internationalized application allows a developer to create and maintain a single code base for that application. In regional or national markets, developers known as **localizers** adapt the application for their particular languages and cultures. A localizer usually changes the data or text of an application's human interface, but the source code for the application remains unchanged.

Text Objects

For text displayed in the human interface, the Toolbox supports all modern languages and many ancient ones through a system-wide data type called a **text object.** A text object is the fundamental unit of text interchange in Mac OS 8. For example, when specifying a filename to the File Manager, an application passes the filename in a text object; and when an application receives text input from the user, the operating system delivers the input in the form of text objects. (In Mac OS 8, text input can come from sources other than the keyboard, such as from a microphone via speech-recognition software or from a graphics tablet via handwriting-recognition software.)

A developer uses text objects for, among other purposes, specifying text displayed as part of an application's human interface. Examples include menu items, window titles, caption text, and control titles. A developer stores these text objects in resource files. Resource files can be edited by localizers without requiring changes to the application's executable code.

Text objects allow developers to manipulate text without dealing with the details of various text-encoding systems. A text object consists of

- a text-encoding specification
- a locale identifier
- the text itself
- optional annotations that may apply to ranges of the text

A **text-encoding specification** identifies the text-encoding system used to express the text. A **text-encoding system** is a computer representation for one or more character sets used by one or more languages and regions. For instance, Unicode is a 16-bit text-encoding system that provides a code for every character in every major writing system. A **locale identifier** encapsulates an International Standards Organization (ISO) language code (which specifies the language in which the text is to be represented) and an ISO region code (which specifies the geographical region for languages that vary by region). For example, a locale identifier may specify English as the language used in a text object, and Great Britain as the geographical region for the language. This regional identification allows different text services and utilities, such as spell checkers and hyphenation dictionaries, to associate British language variations with the text in the object.

A text object is opaque—that is, developers can't see or directly access its contents from their programs. Instead, developers must use a programming interface to gain access to this data. (By supporting data opacity, text objects borrow from object-oriented programming design. However, text objects aren't fully object-oriented because they don't support inheritance, subclassing, or polymorphism.) The programming interfaces provided by Mac OS 8 allow developers to prepare and handle text objects in multiple languages without worrying about the low-level details of the encoding systems used for the text.

Because they enclose a text-encoding specification and language and region information along with the text, text objects make it possible for software that did not create text to process it correctly in any environment that uses multiple writing systems. For example, the user interface elements of an application localized for Hebrew will be depicted in the Hebrew language on a U.S. Mac OS Roman-language system if the Hebrew character set and corresponding glyphs used to represent the text are available on that system.

Text objects also provide a way for developers to associate arbitrary annotations with text. For example, text objects might be annotated with specifications about what color to use when displaying the text, pronunciation hints for text-to-speech conversion, and phonetic information that allows file sorting in languages, such as Japanese, that don't use phonetic alphabets.

MAC OS HERITAGE

WorldScript

WorldScript is a Mac OS programming model for developing international applications. Encompassing technologies that became available in System 7.1, WorldScript defines an approach to programming and software design that includes the use of human interface design strategies and specific programming interfaces supplied by the operating system.

For System 7 developers, the WorldScript I and II system extensions are part of the WorldScript programming model. The WorldScript I extension controls the display, manipulation, and printing of 1-byte complex text-encoding systems for such languages as Hebrew and Arabic. (Support for the simpler 1-byte Roman text-encoding system is built into System 7.) The WorldScript II extension supplies this type of support for 2-byte text-encoding systems—like Chinese and Japanese.

WorldScript doesn't handle all the issues connected with multiple text-encoding systems. For example, a text string contains the data that represents the intended text in a certain language, but a text string does not specify which text-encoding system is being used. Encoding data has to be maintained outside of the text string, a solution that increases the software's complexity and necessitates changing the source code if a developer wants to support a different text-encoding system. By storing the details of text encoding with the text itself, a text object simplifies a developer's efforts.

International Text Input and Display

An internationalized application allows a user to enter and edit text in any language. A single document might contain text in more than one language, so internationalized applications support a mix of languages within the same document. A user might type text in German, then switch to Japanese. Mac OS 8 text-handling services help developers implement these capabilities. Examples of these services include

- Support for speech and handwriting input methods of text entry. It's usually much quicker for a user to enter Asian-language characters by speaking them than by using the keyboard. Multiple input methods may even be used simultaneously.

- A font architecture that allows font formats for multiple languages to coexist easily on a user's system. For example, one font scaler can be used to render characters in Roman languages and another to render characters in Asian languages.
- The internationalized QuickDraw GX graphics system for formatting and displaying text. QuickDraw GX has built-in Unicode support and works with nearly every language—even multiple languages on the same line of text.

See **Chapter 15** for more information about the Mac OS 8 Open Font Architecture and the benefits of using QuickDraw GX to create internationalized applications.

Other International Development Technologies

In addition to text objects, the Human Interface Toolbox supports left-growing windows, dialog boxes that automatically resize themselves around text (which tends to shrink or grow in translation) and other features that address specific international market needs. Mac OS 8 also offers other services for creating applications for international markets. Programs can, for example,

- store and retrieve international preferences and data for a user's workspace
- call various string-comparison functions to compare, order, and search text strings in all languages
- convert text between different text-encoding systems
- use number-formatting and conversion services for any language
- employ date-and-time services that support different calendars (such as the Japanese Imperial Calendar and the Gregorian calendar) and different date and time formats

Compatibility Notes

WorldScript

System 7 application use of WorldScript technologies, including WorldScript I, WorldScript II, the Script Manager, and the Text Services Manager, is fully supported by Mac OS 8. In Mac OS 8, WorldScript I and WorldScript II are implemented as a shared library that's available on every user's system.

Design Extensibility

Developers can use windows, menus, controls, and other human interface objects without modification, or developers can extend these features to support specialized application needs. It's likely that only a small percentage of developers will wish to alter the default behavior or appearance of human interface objects. Nearly all custom user interface features supplied by developers in System 7 applications are implemented as standard features in Mac OS 8. These features include slider controls, tear-off menus, and modifier keys in menus. However, a developer who wants to extend or customize the standard human interface objects has three choices:

- ▶ Add standard panels to a standard embedding panel.
- ▶ Subclass human interface objects.
- ▶ Subclass imaging objects.

Embedding Standard Panels

The easiest way for a developer to create custom human interface objects is to add standard human interface objects to a standard embedding panel. For example, a developer might need to implement an editable text panel that allows the user to change the text only when a certain checkbox is checked. A developer can take these steps to create this arrangement:

1. Instantiate an embedding panel, a checkbox panel, and an editable text panel.
2. Add the checkbox panel and editable text panel to the embedding panel as subpanels.
3. Set a state-change function for the checkbox panel that enables the editable text panel whenever the user selects the checkbox. When the user deselects the checkbox, the state-change function disables the editable text panel.

A developer can store such an arrangement in a resource for the embedding panel.

Subclassing Human Interface Objects

A developer can use System Object Model techniques to subclass custom human interface classes from any standard human interface classes. For example, a developer can create a custom editable text panel that accepts numbers but not letters by subclassing the standard editable text panel class and overriding two of its methods.

To change the appearance of objects defined by a standard human interface class, a developer can subclass it and use Appearance Manager drawing routines to create a new theme-compatible appearance. A developer can use this approach for a wide range of customizing, from making minor adjustments to the appearance of a push button to creating a completely new human interface object.

To create an entirely new human interface object, a developer must subclass the human interface object superclass and build an entirely new object with Appearance Manager drawing routines.

Subclassing Imaging Objects

Using standard System Object Model techniques, a developer can subclass the imaging objects class. Such a subclass can support custom image types for control titles, window titles, menu items, list items, and so on. For example, if an application uses a proprietary graphics format, a developer can subclass the imaging object superclass to implement an imaging object that draws images in the developer's proprietary format. This custom imaging object can then be used with any human interface objects that draw images.

SUMMARY

The Human Interface Toolbox provides a complete programming model for creating an application's human interface. From the user's point of view, a typical Mac OS 8 application displays such human interface objects as

- ▶ windows, alert boxes, and dialog boxes that present data and various choices about manipulating data
- ▶ controls that let the user perform actions or manipulate application settings with a variety of input devices
- ▶ menus that let the user choose from lists of choices or commands

The Human Interface Toolbox supports customizing by individual users. For example, multiple users of one computer can set up their own computing environments—including the details of system-wide appearance, application preferences, the organization of the desktop, and level of complexity—and let the computer handle the details of switching between one environment and the other.

Human interface objects are derived from a single abstract superclass. Two subclasses, one for windows and another for panels, inherit from this abstract superclass. All human interface objects that can be embedded inside windows are derived from the panels class. Without any special programming effort on the part of developers, windows and panels track user input and, according to user and system activity, activate and deactivate themselves, draw themselves using the designs of the current theme, and perform other behaviors automatically.

A developer doesn't need to use an object-oriented language such as C++ to use human interface objects. SOMobjects for Mac OS lets developers code in a variety of object-oriented and procedural languages when using the Human Interface Toolbox. With only minimal knowledge of object-oriented programming techniques, a developer can use the standard human interface objects to create a feature-rich human interface for an application. For example, a C language programmer can use standard human interface objects and create new types of objects by combining standard panels within embedding panels.

Developers who need to customize the human interface further can use object-oriented programming techniques that reliably extend the Toolbox while simultaneously maintaining compatibility with future versions of Mac OS 8. Because the Toolbox implementation is based on SOMobjects for Mac OS, Apple can make future enhancements to human interface features, and these enhancements will be inherited automatically by all applications using the Toolbox.

PLANNING A PRODUCT FOR MAC OS 8

If you're a developer, you can take the following steps to prepare products that take advantage of the Human Interface Toolbox:

1. If you've already created a program for any platform, or if you're currently creating one, separate those portions of your code that handle user interaction from the rest of your application. This will make it easier to incorporate new Human Interface Toolbox features, make your application scriptable, and give your application multiple threads of execution that take advantage of the Mac OS 8 multitasking capabilities.
2. If you're currently designing a new application, keep in mind the needs of the various cultures to which you might wish to market your product one day. To familiarize yourself with this kind of cultural sensitivity, you can read Apple's *Guide to Software Localization* and the sections in the *Macintosh Human Interface Guidelines* that pertain to international software, especially Chapter 2, "General Design Considerations."
3. Don't assume that the alert box or dialog box backgrounds will be white. The Mac OS 8 human interface supports a variety of background colors.

If you've already created a System 7 application, you can take the following measures to make it compatible with Mac OS 8 and make it easier to modify to take advantage of new Mac OS 8 features:

1. Make the portion of your code that handles human interface activity scriptable. This will make it easier to revise your application to use the Human Interface Toolbox programming interface, to support the new event model for Mac OS 8, and to separate your application into different threads of execution.
2. For floating windows, use the standard floating-window definition introduced in System 7.5. This window definition works correctly on Mac OS 8 and inherits the Mac OS 8 appearance.

3. Remove from your code any assumptions about the precise locations of human interface elements such as close boxes, zoom boxes, and window titles within the noncontent areas of windows or dialog boxes.
4. Use routines defined by the operating system to manipulate interface elements instead of directly modifying their data structures.

13 Assistance Services

The Assistance Services provide developers with comprehensive and flexible facilities for delivering help appropriate to users' various goals and skill levels. This help can range from the automation of complex operations on the user's behalf to the display of useful descriptions of onscreen human interface features.

Developers can use the Assistance Services to automate as much of the user's work as possible. A program can interview the user to gather information relevant to a user's work goals and then carry out the actions necessary to accomplish those goals. These actions can be deferred until user-specified times, whether or not the user is actually at the computer.

The Assistance Services learn how the user performs computer operations. When a more efficient way to perform an operation is available, the operating system tells the user so. This type of assistance takes the burden of searching for productivity-enhancing tips away from users, allowing them to learn these tips under the very circumstances in which they can be applied.

To be most useful, onscreen assistance must be comprehensive and flexible in its approaches to delivering help appropriate to users' goals and skill levels. So, for users who need only quick access to information about performing tasks and using application features, the Assistance Services also supply facilities for presenting interactive tutorial and reference information.

KEY TERMS AND CONCEPTS

- ▶ An **expert** is a small program that interviews the user to gather information about goals and preferences. The program uses this information to help the user carry out a complex or seldom used operation, such as setting up a computer or initially connecting to the Internet.
- ▶ The **Interview Manager** is a service used by a program, such as an expert, to obtain information from the user so that the program can automate or delegate operations.
- ▶ The **Trigger Manager** tracks events or states—such as specific times, time intervals, or other programmatically determined conditions—that invoke delegated tasks.
- ▶ A **delegated task** is a user-scheduled operation performed automatically when a programmatically determined set of circumstances occurs.
- ▶ The **Notification Manager** is used by a program, such as an expert, to inform users about the status of program operations, including delegated tasks.
- ▶ Applications use the **Tip Manager** to offer user suggestions about making more efficient use of program features.
- ▶ **Apple Guide** is an onscreen help system that explains concepts or guides users through the steps of an operation.
- ▶ A **help balloon** is small window containing explanatory information about the onscreen item to which the cursor points. Help balloons look like the dialog bubbles in comic strips. Developers use the Help Manager to provide help balloons to users.

MAC OS HERITAGE

Balloon Help and Apple Guide

Users of System 7.5 are already familiar with Balloon Help and Apple Guide. Balloon Help first appeared in System 7.0, and System 7.5 introduced Apple Guide. These initial facilities helped users learn to manipulate applications. In addition to Balloon Help and Apple Guide, Mac OS 8 supports more active forms of user assistance, where the goal is to make the computer a collaborating partner that actively helps users take greater advantage of their application capabilities.

MAJOR POINTS OF INTEREST

To help users take the fullest possible advantage of their applications, developers provide users with various forms of assistance. One form is usually a printed guide that explains a program's uses and capabilities. But many people don't read manuals—at least not in their entirety. And even after they do read a manual, users often forget instructions that are very complex or are seldom employed. Programs that provide user assistance directly onscreen—when and where users need it—help people use their computers more capably and productively. Well-designed user assistance also lowers customer support costs for developers.

Developers use the Assistance Services to provide onscreen instructional information such as tutorials and reference materials. The services include Apple Guide, the Help Manager, the Interview Manager, the Trigger Manager, the Notification Manager, and the Tip Manager.

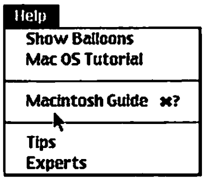
With Apple Guide, developers provide users with an interactive way to learn product features and to automate the use of many features. The Help Manager allows developers to display quick-reference descriptions of their onscreen features.

The most direct way for developers to assist users is to supply them with experts—small programs that interview users and consult with them about performing complex, seldom used, or difficult-to-remember operations. Mac OS 8 provides a variety of ready-made experts. Developers can create their own experts for a variety of purposes. For example, an application might include an expert that helps the user create a digital video.

For developers, experts provide a way for their products to perform complex operations without first requiring the user to learn complex details about its human interface. For consultants and instructional designers, experts provide a way to automate specialized tasks involving multiple off-the-shelf applications. When implementing an expert, a developer typically uses the Interview Manager to collect information about the user's goals. To schedule the execution of actions that fulfill users' goals, the developer uses the Trigger Manager. The Trigger Manager allows experts to assist the user even when the user isn't present at the computer. Although the user might be away, for example, an e-mail expert can automatically dial an online service and check for mail every day, twice every day, or every hour as scheduled by the user and administered by the Trigger Manager.

To inform users about the status of delegated tasks, a developer uses the Notification Manager. User notifications can take such forms as sounds, icons that blink at the top of the screen, reports in a log file, and onscreen alert boxes containing short messages. For example, an e-mail server might display a small dialog box informing the user that new mail has arrived or that no connection was made to an online mail service.

FIGURE 13.1 The Help menu

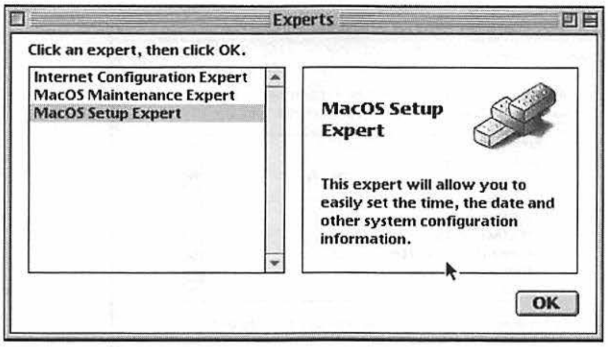


These managers—the Interview Manager, Trigger Manager, and Notification Manager—are also available for use within any application. That is, developers don't need to create stand-alone experts to perform user interviews, undertake work at user-scheduled times, or offer user notifications. Instead, developers can use any combination of Interview Manager, Trigger Manager, and Notification Manager services in any type of application.

Assistance is also offered to the user by way of tips, which are instructions for making more efficient use of application and Mac OS features. Part of every workspace includes preferences for the invocation and display of user tips, which the Tip Manager displays under the very circumstances where users ought to employ these tips. For example, the Tip Manager tracks user activities related to closing a window: choosing the Close command from the File menu, clicking the close box of a window, and using the Command-W keyboard equivalent. When the Tip Manager detects that the user has repeatedly used the least efficient technique—namely, choosing the Close command from the File menu—the Tip Manager can alert the user to the availability of a shortcut—namely, use of the Command-W keyboard equivalent.

Users can find assistance by looking at the Help menu, shown in Figure 13.1. The Help menu appears in the menu bar of all applications. (The Help menu replaces the menu represented by the help icon in System 7.) Assistance is also available to users through **contextual menus.** Developers use the Human Interface Toolbox to supply their applications with contextual menus. The user invokes a contextual menu by moving the cursor to an item on the screen, then holding down a modifier key while pressing the mouse button. A menu of commands relevant for the current context appears onscreen. For example, a user might select text and then open a contextual menu containing commands to cut, copy, and clear the selected text, and to apply various character styles to the text. The menu might also contain commands to invoke help balloons, Apple Guide sequences, or experts.

FIGURE 13.2 An experts access panel



This expert will allow you to easily set the time, the date and other system configuration information.

EXPERT ASSISTANCE

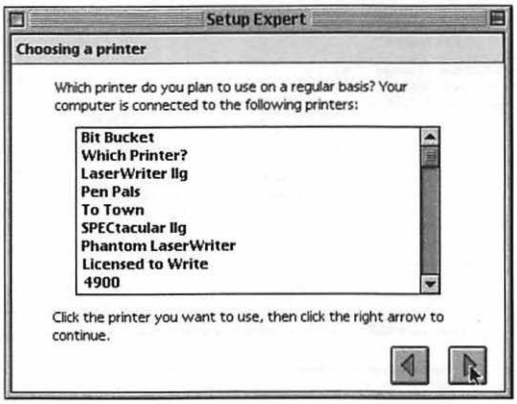
A developer can create experts in the form of small programs that consult with users about accomplishing goals—for instance, editing a digital video, backing up a disk, or setting up a computer on a network. An expert interviews the user to gather detailed information necessary for accomplishing a particular goal. The expert then automates as much of the work as possible on behalf of the user. The expert can perform the work immediately or allow the user to schedule the work to be performed later. The expert can also notify the user of the status of this work.

An expert allows the user to focus on accomplishing specific goals instead of learning to use an application. By posing questions to solicit information from the user, an expert interacts with the user through a more natural interface than that provided through the standard windows and menus of an application. For example, the developer of a disk backup application can create an expert that helps users regularly archive their files. Such an expert would ask a user when and how often to back up files. The expert would then call the application to perform the backups automatically, according to a user's wishes—say, every Friday at 6:00 PM—and the expert would inform the user about the outcome of each weekly backup.

Experts become available to users in several ways. An application can list experts in its Help menu and contextual menus, and experts can be invoked automatically during an application's installation or every time an application is launched.

When the user selects the Experts command from the Help menu or a contextual menu, Mac OS 8 presents an experts access panel similar to the one shown in Figure 13.2. For Mac OS 8 to display an expert in this panel under

FIGURE 13.3 An interview panel from an interview sequence



Choosing a printer

Which printer do you plan to use on a regular basis? Your computer is connected to the following printers:

Bit Bucket
Which Printer?
LaserWriter IIg
Pen Pals
To Town
SPECtacular IIg
Phantom LaserWriter
Licensed to Write
4900

Click the printer you want to use, then click the right arrow to continue.

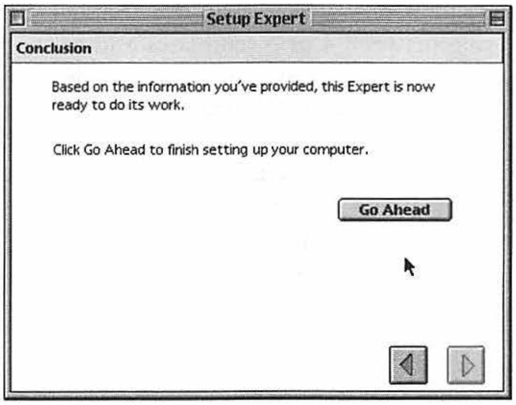
the appropriate circumstances, the expert registers itself with the operating system. When the user selects an expert from the scrolling list on the left side of the panel, a description of the expert appears on the right side of the panel. To initiate the selected expert, the user clicks a button in the lower-right corner of the panel.

After the user clicks this button, the selected expert initiates an **interview sequence**—an interactive conversation between the user and the expert. **Interview panels** contain questions for the user. Figure 13.3 shows one of the interview panels for the Setup Expert supplied with Mac OS 8. Responding to interview panels such as this, the user makes choices concerning the operations that the expert is designed to perform. An expert can also use interview panels to ask the user whether to defer the operation for later execution, and if so, when or under what circumstances the operation should be performed. The last panel presented by an expert includes a Go Ahead button. A user clicking Go Ahead initiates or schedules the actions that the expert is designed to perform. Figure 13.4 illustrates the last interview panel presented by the Setup Expert.

A **compiled program** is created in source code, then transformed by a compiler and linker into executable code. An **interpreted program** is translated for execution by a separate program called an interpreter.

A developer can create an expert as a compiled program or as an interpreted program, such as an AppleScript script. To present interview panels consistent with other Mac OS 8 programs, an expert uses the programming interface provided by the Interview Manager. To delegate operations for automatic execution under user-specified circumstances, an expert uses the programming interface provided by the Trigger Manager. To inform the user about the status of delegated tasks, an expert uses the programming interface supplied by the Notification Manager.

FIGURE 13.4 The last interview panel in an interview sequence



Based on the information you've provided, this Expert is now ready to do its work.

Click Go Ahead to finish setting up your computer.

MAC OS HERITAGE

AppleScript

Scripting languages are designed to be easier to learn and use than complex programming languages like Pascal or C. A **script** is a series of statements—written in a scripting language—instructing a computer system to perform various operations. Scripts are translated for execution by interpreter programs.

AppleScript is the scripting language built into the Mac OS since System 7.1.1. It allows developers and technically adept users to automate routine or highly complex tasks and to integrate off-the-shelf applications. AppleScript is based on the Open Scripting Architecture (OSA), which defines a standard mechanism for coordinating and automating multiple programs with scripts written in a variety of scripting languages. An OSA-compliant script coordinates and automates programs by sending them commands packaged as Apple events. At the request of an expert written as an AppleScript script, for example, an interpreter program sends Apple events directing the operations of scriptable programs. To be considered "scriptable," a program must understand and respond to the Apple events sent by scripts.

THE ARCHITECTURE OF THE EXPERT ASSISTANCE SERVICES

The services offered by the Interview Manager, Trigger Manager, and Notification Manager are available to any type of program—that is, they aren't reserved for exclusive use by experts. A program other than an expert can incorporate some or all of these services. For example, a word-processing application might use the Interview Manager to present the user with a short list of questions relevant to creating a resumé—for instance, questions related to choosing between 4 or 5 templates and to supplying personal information appropriate for the chosen template. Under this scenario, the application wouldn't use the Trigger Manager to delegate any future operation. A file-compression program, by comparison, may have a user interface that doesn't require an interview sequence, but the program might use the Trigger Manager for scheduling automated file-compression operations. A source code compiler, as a final comparison, might not need to perform interviews or necessarily delegate operations but might use the Notification Manager to alert the user about the conclusion of a code compilation or about any programming errors that halted a compilation.

User Interviews

To gather information from a user, any type of software—an expert, an OpenDoc part editor, or a full-featured application—can use the programming interface provided by the Interview Manager. The Interview Manager supplies programs with a standard set of human interface elements for performing interviews. This interface consistency allows users to quickly gain skills that can be transferred among different programs.

After receiving interview responses from the user, a program immediately performs the appropriate operations or schedules them for later execution. For instance, when using an interview sequence to help the user prepare a resumé, a word-processing application might immediately supply a resumé template based on the choices and preferences supplied by the user during the interview. An expert designed to help the user manage e-mail, by comparison, might delegate several e-mail programs to dial their respective online services and download mail at user-specified times. This expert might direct a file-decompression program to expand any compressed files that had been downloaded. The expert might also notify the user of the arrival of new mail. These delegation and notification capabilities are discussed in the next two sections.

Delegation

The Assistance Services support delegation—the automation of user tasks to be performed at later times. With delegation, the user needn't be at the computer to be productive. For example, a disk-backup program can back up a user's files when the user isn't at the computer.

Any type of program can use the Trigger Manager to incorporate delegation. Experts usually invoke the Interview Manager to ask the user when to schedule operations, but delegation doesn't require use of the Interview Manager, nor is delegation restricted to experts. A file-compression program, for example, might employ a simple dialog box instead of an interview to gather user preferences, and the program might then use the Trigger Manager to automate the compression of data files that have aged a user-specified number of days, weeks, or months.

Such a delegated task can be performed in the background. A delegated task is scheduled by the user to be performed automatically upon the occurrence of a programmatically determined set of circumstances. There are three major aspects of a delegated task:

1. The **trigger condition.** This is the particular state of the computer that should cause the execution of a delegated task. For example, the condition could be one minute after every hour, whenever a disk volume is mounted, or whenever a file has been modified within a particular folder.
2. The operations necessary to perform the delegated task. For example, a delegated task could involve dialing an online service, downloading new mail, and expanding any compressed files that accompany the mail.
3. The **user notification.** This is the way that the user learns of the outcome of the delegated task. For example, a program might display a dialog box informing the user that new mail has arrived or that no connection was made because the number for the online service was busy.

Trigger Conditions

While waiting to perform delegated tasks, a program doesn't need to be running; trigger conditions automatically cause the program to start up and perform its operations. Trigger conditions can be based on specific moments in time, time intervals, or a variety of other events or states that can be determined programmatically. A program registers trigger conditions with a trigger module. A **trigger module** is a shared library containing information about trigger conditions. Whenever circumstances match those specified in a trigger condition, the trigger module sends an Apple event to the registered program. The program then undertakes the delegated task associated with that trigger condition.

The Trigger Manager supplies developers with several standard trigger modules, including

- ▶ a time trigger module for conditions based on the time of day, the date, the day of the week, elapsed time, and any combination of these
- ▶ a volume-mounted trigger module for conditions based on the moment volumes are mounted, such as when a CD-ROM or other removable medium is inserted into an attached disk drive or when a remote volume is mounted across a network
- ▶ a folder-changed trigger module for detecting additions, removals, and edits to files in specified folders

Trigger modules are SOM objects, which were described in Chapter 9. Using SOMobjects for the Mac OS, developers can create their own trigger modules and market them to users and other developers.

So that users can select trigger conditions through a consistent human interface, the Trigger Manager supplies a standard trigger-picking panel for choosing trigger modules. Developers include this panel in interview panels or embed them in their application windows or dialog boxes.

An expert or any other type of program can register a user-selected trigger condition. For example, a disk-backup program could register 7:00 P.M. every Friday as a trigger condition. At that moment every week, the time trigger module informs the disk-backup program that the trigger condition has been met; the disk-backup program, in turn, performs its automated file backups then.

Delegated Tasks

A program has to provide code that actually performs the action delegated by the user—for instance, dialing an e-mail service every other hour, backing up disks every weekend, or compressing files when they reach a certain age. As you've read, this code can be implemented in a program that registers trigger conditions with the Trigger Manager. The program then receives Apple events informing it of occurrences matching those specified in these trigger conditions, and the program responds by performing its delegated operations.

Or this code can be implemented, at least partly, in an AppleScript script, which the operating system starts when the script's trigger conditions are met. Anyone who writes AppleScript scripts can associate a trigger condition with a script stored in a special folder on the system. When a circumstance defined as a trigger condition occurs, the operating system runs the appropriate script. These scripts generally send Apple events directing other programs to perform the operations delegated by the user. For example, an expert might use the Interview Manager and the standard trigger-picker panel to determine that the user would like an e-mail program to call an online service at 8:00 A.M. every morning. Every day at this time, the operating system could automatically run a script that would use Apple events to start the e-mail program and to instruct it to dial the online service and check for mail. This scripting ability is not restricted to use by experts or other compiled programs; anyone familiar with the AppleScript language can create delegated tasks that launch and automate scriptable applications.

Notification

Because delegated tasks may be performed while a user is working with another application or even while a user is away from the computer, it is important that programs report the outcome of these actions—that is, whether the actions were successful or why they failed. Programs can also send user notifications about the outcome of any type of action, because user notifications aren't reserved exclusively for delegated tasks.

To assist programs in notifying users of the status of any type of operation, the Notification Manager supplies several standard notifier modules. Like trigger modules, notifier modules are implemented as SOM objects. **Notifier modules** are shared libraries containing information about what types of notifications users want to receive. Developers incorporate a notifier-picker panel into interview panels or into their application dialog boxes or windows. This panel lets users select among several choices:

- to see a dialog box containing a notification
- to hear an alert sound
- to see a program icon blinking in alternation with the Application menu icon (thereby alerting the user to bring the program to the foreground)
- to receive a report in a log file

Developers can also create their own notifier modules. For example, a developer might choose to supply a notifier module that pages the user.

A program saves notification preference data solicited through the notifier-picker panel. After performing an operation, the program (which may also be an AppleScript script) passes this data back to the Notification Manager along with information about the outcome of the operation. The Notification Manager then prepares and sends the appropriate type of notification informing the user of the outcome of that operation.

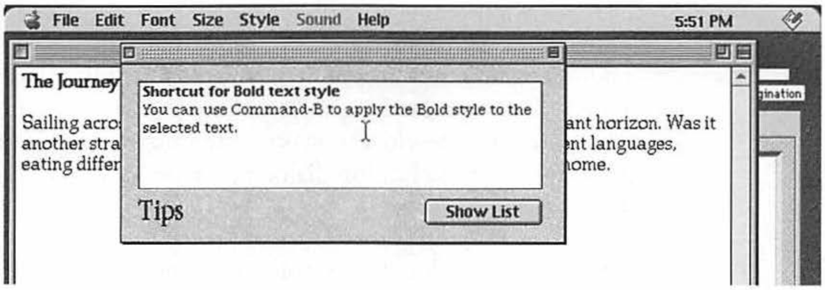
HELP INFORMATION SERVICES

Onscreen assistance must be comprehensive and flexible in its approaches to delivering assistance appropriate to users' goals and skill levels. Experts and other programs performing automation and delegation fulfill many user help needs, but not all of them. Often users simply want quick access to information about performing application tasks and using application features. Mac OS 8 provides several services through which applications supply helpful information to their users.

Reference information is appropriate in some situations, and tutorial information is more appropriate in others. A user might simply need to know what an onscreen element is and what it does; for example, a user might wish to know "What happens if I click the Eject button in this dialog box?" That user simply needs a brief description of the Eject button. The features of the Help Manager allow developers to easily provide such descriptions.

Later, the user might wish to know how to perform a task that involves multiple controls and menu commands—for example, a user might wish to

FIGURE 13.5 A tip for applying text styles



Shortcut for Bold text style
You can use Command-B to apply the Bold style to the selected text.

know "How do I switch my display between color and grayscale?" That user needs onscreen instruction. Developers use Apple Guide to provide instructional information to users. Developers can even automate operations for the user through Apple Guide.

Other times, the user might benefit from instruction related to using an application more efficiently. The Tip Manager allows the application to display helpful suggestions at appropriate times.

The sections that follow explain how developers provide users with tips through the Tip Manager, instruction and reference through Apple Guide, and descriptive help balloons through the Help Manager.

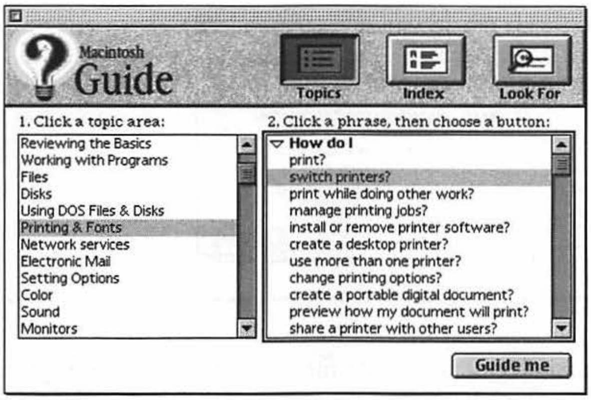
Tips

As you will see, both Apple Guide and Balloon Help require users to seek out information. However, with the Mac OS 8 **tips** feature, instructions for making more efficient use of application features present themselves to users who can benefit from the information. Even experienced users find themselves surprised from time to time to learn new shortcuts. By watching their colleagues (or their children), users continue to learn operations that save time. Applications that support the Tip Manager display helpful suggestions to users who can benefit from them.

On behalf of applications, the Tip Manager maintains a database of user actions. When a user repeatedly performs an action for which there is a more efficient alternative, the Tip Manager alerts the user. For example, suppose a user who wants to learn about tips repeatedly applies the boldface character format to text by selecting a command from the Style menu. Mac OS 8 detects this as an inefficient technique and displays a suggestion for using a keyboard shortcut to apply the boldface style, as shown in Figure 13.5.

Users decide whether tips are displayed and how the Tip Manager signals their presence. To avoid overwhelming the user with information, the Tip

FIGURE 13.6 An Apple Guide access window



1. Click a topic area:

Reviewing the Basics
Working with Programs
Files
Disks
Using DOS Files & Disks
Printing & Fonts
Network services
Electronic Mail
Setting Options
Color
Sound
Monitors

2. Click a phrase, then choose a button:

How do I
print?
switch printers?
print while doing other work?
manage printing jobs?
install or remove printer software?
create a desktop printer?
use more than one printer?
change printing options?
create a portable digital document?
preview how my document will print?
share a printer with other users?

Manager waits until the user repeats an action a certain number of times before informing the user that a more efficient action exists. (Applications can control what that threshold number should be.) The user can then read a description of the recommended tip or shortcut.

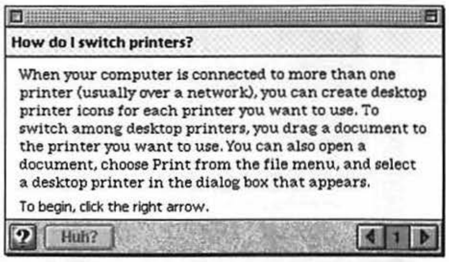
To offer tips with an application, a developer designs the application to add information to the Tip Manager database. Using the programming interface defined by the Tip Manager, the application supplies two tables of information:

- The first table specifies the types of actions the Tip Manager should track and, for each action, the threshold at which the Tip Manager should alert the user. For example, the application may want the Tip Manager to track the number of times the user opens a dialog box to invoke a command that's more accessible from a menu.
- The second table specifies the messages that the Tip Manager should display when each user action listed in the first table reaches its threshold. For example, after the user invokes a command from a dialog box ten times, the Tip Manager could prepare a message suggesting that the user choose the command from its menu instead.

Apple Guide

Apple Guide first became available in System 7.5 as an instructional tool, and it remains a valuable assistance feature in Mac OS 8. The user opens Apple Guide from the Help menu, contextual menus, or from a keyboard shortcut assigned by the active application. An access window for the relevant help information then appears. An Apple Guide access window, such as that shown in Figure 13.6, presents a list of questions, problems, and tasks determined by

FIGURE 13.7 An Apple Guide presentation panel


How do I switch printers?

When your computer is connected to more than one printer (usually over a network), you can create desktop printer icons for each printer you want to use. To switch among desktop printers, you drag a document to the printer you want to use. You can also open a document, choose Print from the file menu, and select a desktop printer in the dialog box that appears.

To begin, click the right arrow.

the designer of the guide file. A user clicks the Topics, Index, and Look For buttons to view potential topics and select the one of interest.

Once the user selects a topic and clicks the Guide Me button at the bottom of the access window, Apple Guide closes the access window and displays a presentation panel like the one in Figure 13.7. When the user clicks the right and left arrows in the lower-right corners of the panels, Apple Guide leads the user through explanations of the selected topic. Because Apple Guide panels are movable and float on top of other application windows, the user can carry out the instructions in the panels while working with applications.

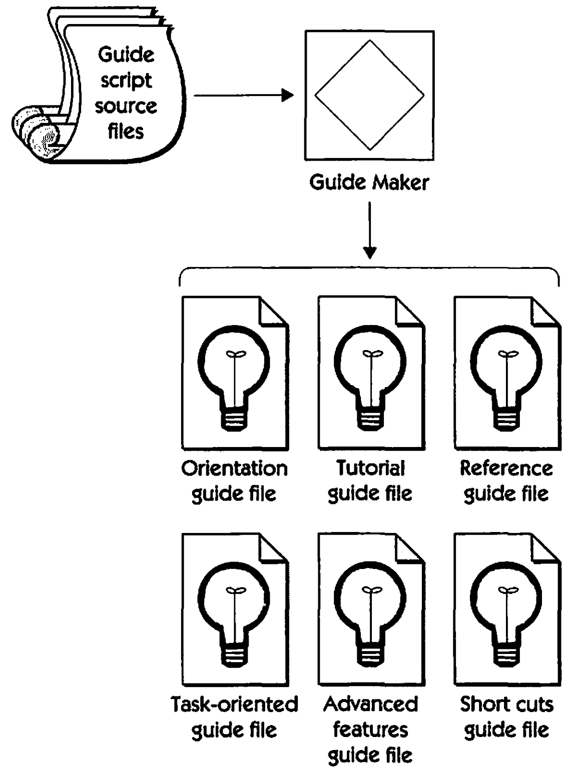
When appropriate in the context of a particular panel, a coachmark can direct the user to areas where input is required. **Coachmarks** are graphical elements that point to, circle, or otherwise indicate items on the screen. For example, suppose a presentation panel instructs the user to open the File menu. To indicate what the user needs to do, Mac OS 8 draws a circle around the File menu. In addition to the circle, an underline, an arrow, and an "X" character are available as coachmarks. Developers can use coachmarks anywhere in their applications. In Mac OS 8, the use of coachmarks isn't restricted to Apple Guide, as it is in System 7.

Apple Guide is instructive and answers the question "How do I accomplish this task?" Developers can use Apple Guide to provide

- ▶ orientation to an application's entire help system
- ▶ task-oriented instructions for using application features
- ▶ tutorials that guide users through focused learning paths
- ▶ instructions for using advanced or specialized features
- ▶ reference material commonly found on quick reference cards

Apple Guide onscreen help information is stored in **guide files.** To create guide files, a developers first creates source files using a scripting language called **Guide Script.** The developer then uses a utility called **Guide Maker,**

FIGURE 13.8 Creating guide files



available from Apple Computer, to build and test the guide files. Figure 13.8 illustrates the steps developers take to create guide files.

Guide file development isn't limited to commercial application developers. In-house programmers, corporate trainers, and computer consultants also create guide files that help users. For example, a trainer can create a guide file that teaches employees how to enter information into a company's databases.

A developer can add guide files to an application without changing its source code. The user opens these guide files by selecting them from the Help menu. A programmer can also call the Apple Guide programming interface from within the application to start Apple Guide and to open, close, and control guide files. In an application dialog box, for example, a developer might place a button that allows the user to open a relevant guide file.

In Mac OS 8, guide files can send Apple events to applications; these Apple events, in turn, can describe actions that applications should perform. By using Apple Guide in conjunction with a scriptable application, developers can provide a level of assistance that goes beyond mere instructional help.

This level of help allows users to automate many operations. Automated Apple Guide assistance is a good way to help users perform a task that's hard to learn, easy to forget, or too time-consuming to undertake manually. To help the user complete such a task, a guide file can streamline the steps, taking the user directly to key interface elements when user input is required and directing applications to automatically perform as many of the steps as possible.

For example, suppose a user wants assistance preparing and sending e-mail. A guide file could send Apple Events to an e-mail application instructing it to launch itself and to open a mail-editing window for the user. The guide file could then use a presentation panel and a coachmark prompting the user to type the text of the message into the application's mail-editing window. After another panel prompts the user to type the subject of the message, the guide file, using Apple events to interact with the e-mail application, could automatically display the possible recipients and prompt the user to choose one. Finally, the guide file could direct the application to send the e-mail after the user clicks a Send button in a final Apple Guide presentation panel.

COMPATIBILITY NOTES

Guide Files

Mac OS 8 fully supports the guide files created for Apple Guide in System 7.5. Apple Guide presentation panels in System 7.5 can contain text, pictures, buttons and other controls, recorded sounds, pictures, and QuickTime movies. Presentation panels in Mac OS 8 add support for text input areas as well.

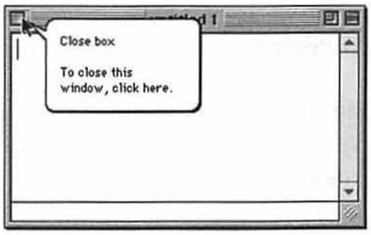
Help Balloons

Help balloons provide brief online descriptions of human interface elements, allowing users to learn the actions, behaviors, and properties of onscreen features. In a very concise manner, help balloons answer one of these questions for the user:

- What is this?
- What does this do?
- What happens if I click this?

The user turns on Balloon Help from the Help menu or contextual menus. For example, a new user might not understand the purpose of the close box in a window. When the user turns on Balloon Help and moves the cursor to the close box, as shown in Figure 13.9, the screen displays a help balloon describing this element. All normally available application features remain active.

FIGURE 13.9 A help balloon



That is, help balloons simply display information; they don't interfere with typing or use of the mouse.

Help balloons were introduced in System 7 and remain fully supported in Mac OS 8. When a user turns on Balloon Help, the Help Manager tracks the cursor and automatically displays help balloons for menus, window frames, the contents of static windows (like dialog boxes), and non-document Finder icons. Applications simply provide the text or pictures that appear in these balloons. To provide help balloons for the contents of dynamically changing windows (such as those with scroll bars), applications must track the cursor and use the Help Manager programming interface to display help balloons.

COMPATIBILITY NOTES

The Help Manager

Mac OS 8 fully supports the help balloons implemented by software that incorporates the help resources and programming interface of the System 7 Help Manager.

SUMMARY

With the Assistance Services, developers can provide several kinds of user assistance, including

- help balloons, which display information that briefly describes and explains the use of human interface elements
- Apple Guide, which presents interactive instruction about application features and helps automate the use of application features

- tips, which a user can elect to see whenever Mac OS 8 detects that they could benefit the user
- experts, which interview users about work goals and then perform much of the work necessary to achieve those goals

The architecture supporting experts also allows developers to directly incorporate facilities for interviewing users, deferring execution of operations, and notifying users of the status of program operations.

PLANNING A PRODUCT FOR MAC OS 8

If you are a developer, you can take the following steps to prepare products that take advantage of the Mac OS 8 Assistance Services:

1. Provide one or more guide files for your existing product. If you have a Windows product, the Guide Maker tool can help you convert your Windows help to Apple Guide files.
2. Make your existing application AppleScript scriptable. This will help you to incorporate delegation and automated Apple Guide assistance into the application.
3. Consider which operations your users might like to delegate for later execution, and think about the trigger conditions you might use and the types of notification you might give. These considerations will help you design delegation into your product.
4. Design your next product or revision with an eye toward what activities your users need to perform rather than what features you can present. This approach can help you create experts for your product.
5. Consult your technical support staff to find out which questions your users ask most frequently. Then determine whether tips might address some of the difficulties your users report.
6. Determine whether you should create a stand-alone expert or integrate Mac OS 8 assistance features, such as interviews and notifications, within your application.
7. If you've created a System 7 extension product that automatically performs background actions under particular circumstances, plan how you can reimplement that product as a server program that uses the Trigger Manager.

14

Events

An **event** is a user action or system occurrence requiring a response from a program. Events include keystrokes and mouse clicks from the user, requests from other programs (for example, to print files), or any other activities in the system (for example, the completion of I/O operations).

Events drive task execution in Mac OS 8. Tasks spend little time actually being executed on the CPU. Instead, they're normally blocked, waiting for events to which they can respond. The main task of a cooperative program, for example, stops executing when there are no events for it. It becomes eligible for execution as soon as an event for it arrives. When the microkernel gives that task access to the CPU, the task responds to the even. The task then stops executing, allowing other tasks on the system to efficiently share the CPU.

Mac OS 8 informs programs about events chiefly by sending Apple events. For example, when the user chooses a command from an application menu, the operating system sends the application an Apple event containing the information that the application needs to begin responding. For compatibility with older applications, Mac OS 8 also supports events based on the Event Manager model from System 7.

Key Terms and Concepts

- An Apple event is a data structure used to direct the operation of or communicate information to a program. An Apple event identifies itself and its purpose, names its destination, and contains additional data structures that vary according to the kind of event.
- An **Apple event handler** is a function that extracts pertinent data from an Apple event, performs the action requested by the Apple event, and returns a result.
- A **handler table** contains a group of Apple event handlers.
- The **Apple Event Manager** is an operating system service that allows programs to send and receive Apple events.

Major Points of Interest

Mac OS 8 provides a unified model, based on Apple events, for implementing all forms of event handling. Developers can use this model to implement event handling for both cooperative programs (including OpenDoc parts) and server programs. The essence of the Mac OS 8 event model is simple. When a task runs, it expresses an interest in receiving certain Apple events. It then informs the Apple Event Manager that it's ready to receive. The Apple Event Manager blocks the task until one of these events arrives. This approach maximizes the efficiency of priority-based preemptive scheduling, allowing other tasks to receive processing time when a program doesn't need it.

Apple events are the standard means Mac OS 8 uses to inform programs about the events to which they must respond. OpenDoc parts also receive events in the form of Apple events. When cooperative programs use Apple events with the Mac OS 8 event model, the Process Manager automatically synchronizes their access to the Human Interface Toolbox and other cooperative services.

Chapter 4 discusses how the Process Manager serializes cooperative program access to the cooperative services.

Server programs can use both Apple events and other interprocess communication mechanisms, such as the Microkernel Message Service and the System Notification Service, to handle communications in a fully reentrant manner. Compared to Apple events, these low-level mechanisms can potentially provide slightly better performance, but they entail significantly more development effort to implement.

Apple events, first introduced in System 7, are the primary mechanism for interprocess communication in Mac OS 8. See **Chapter 5** for more information about interprocess communication.

Every Apple event identifies its destination, and the operating system delivers the Apple event to that destination. For example, when the user clicks an inactive window, the operating system sends an Apple event to the main task of the cooperative program that created the window. This Apple event directs the program to make the contents of that particular window active—for example, by allowing the user to manipulate the text or graphics in the window.

Every Apple event contains attributes, called the event class and the event ID, that identify the Apple event and denote its purpose. An **event class** identifies a group of related Apple events. An **event ID** identifies a particular Apple event within a group of related Apple events. For example, an event's class might identify it as one of a group of text-related events, and its ID would identify it as a particular type of text event.

When an Apple event is sent to a program, the Apple Event Manager invokes an Apple event handler provided by the program. The handler is designed to respond to that particular type of Apple event. The handler extracts pertinent data from the Apple event, performs the action requested by the Apple event, and returns a result.

When launched, a program uses the Apple Event Manager to create handler tables that map Apple events to program-defined handlers. When a task is ready to receive Apple events, it calls the Apple Event Manager function AEReceive. This function doesn't return a result unless a handler generates an error or the handler intentionally terminates the call to AEReceive.

The AEReceive function blocks the task that called it until the Apple Event Manager sends an Apple event to that task. The Apple Event Manager delivers only those Apple events for which handlers are available. When an Apple event arrives, the task becomes eligible for execution. The microkernel allocates execution time for the task, and the Apple Event Manager dispatches the Apple event to the appropriate handler. After the task executes its handler, it becomes blocked again by the AEReceive function. This cycle of blocking then executing a task continues until the task terminates or the program quits.

An OpenDoc part editor doesn't call AEReceive. Instead, it calls the OpenDoc method HandleEvent—the same method used in System 7. OpenDoc then delivers Apple events directly to the part editor. By using handlers that respond to Apple events, a part editor adheres to the Mac OS 8 event model.

MAC OS HERITAGE

Event Handling in System 7

In System 7, an application uses a section of code called the **event loop** to processes events. The event loop repetitively requests events from a service called the **Event Manager.** When the Event Manager returns events, they're dispatched to application-supplied event-handler routines. When there are no events for the event loop to handle, the application yields. If there's an event for another application, it gets to execute; otherwise, the yielding application gives other applications a very short period of time to execute background operations. At the end of this period, the application resumes execution, and its event loop calls the Event Manager again to request new events. The application dissipates valuable CPU cycles polling the Event Manager even when there are no events to report.

By comparison, a Mac OS 8 application performs relatively little event processing. Instead, an application—blocked from execution—relies on the Apple Event Manager to dispatch events directly to event-handler routines supplied by the application. The application gets access to the CPU only when the Apple Event Manager calls the application's event-handler routines. In the multitasking environment of Mac OS 8, where multiple programs share access to the CPU, this model makes much more efficient use of the CPU than does the polling approach used in System 7.

EVENT HANDLING

Developers can use the Mac OS 8 event model to direct the operation of all tasks for any type of program. When instantiated as a process, every program has a main task that may spawn additional tasks. To receive events, any task can call the Apple Event Manager function AEReceive. The Apple Event Manager blocks the calling task until an appropriate event arrives. The task then gets an opportunity to execute, and the Apple Event Manager dispatches the event to a handler associated with that task. After the handler responds as part of the task, the Apple Event Manager blocks the task again until the next event arrives.

Any kind of program can use Apple events to communicate among its tasks or with other programs. For example, a statistics application might implement its user interface as a cooperative program, which could send Apple events as necessary to a separate server program that performs statistical calculations. After calling AEReceive, the server program's main task would become blocked until an Apple event requesting a calculation arrived. At that point the server program's handler for performing the calculation would be invoked, and the CPU could perform the calculation preemptively without disturbing the user's interaction with the cooperative program. When it has finished with the calculation, the handler could send another Apple event back to the cooperative program to inform it of the results. The server program could even spawn additional tasks to start handling requests from other programs before the first calculation is complete.

In the cooperative scheduling environment of Mac OS 8, the main tasks of cooperative programs yield execution eligibility whenever there are no events for them. When tasks have no events, AEReceive causes them to yield by blocking them. (System 7 applications yield by periodically calling an appropriate Event Manager function.)

Tasks can also receive messages through alternate interprocess communication mechanisms. For example, the main task of a cooperative program can use the Microkernel Messaging Service to direct the operation of additional tasks spawned by the program. However, Apple events are handy for a developer because they permit the operating system, programs, and scripts to communicate with each other locally and across networks according to a well-established messaging protocol. To use other interprocess communication mechanisms, developers must establish and follow their own conventions for sharing information between tasks. The use of AEReceive greatly simplifies the amount of coding necessary for a program to handle events. AEReceive is the basis of the Mac OS 8 event model, and the rest of this chapter focuses on its use.

COMPATIBILITY NOTES

The Event Manager and the Event Record

Mac OS 8 supports WaitNextEvent, the event record, and other Event Manager functions and data structures for compatibility with System 7 applications. Applications that use Event Manager functions run as cooperative programs in Mac OS 8, but they can't take full advantage of the improved performance and flexibility that the Mac OS 8 event model makes possible. Developers of new applications should adopt the Mac OS 8 event model instead of using the Event Manager.

The OpenDoc environment in Mac OS 8 continues to use the event handling programming interface defined for part editors in System 7. However, developers of new OpenDoc parts should use the Apple event data structure instead of the event record for their event data format. Part editors can then partake of the improved performance and flexibility of the Mac OS 8 event model.

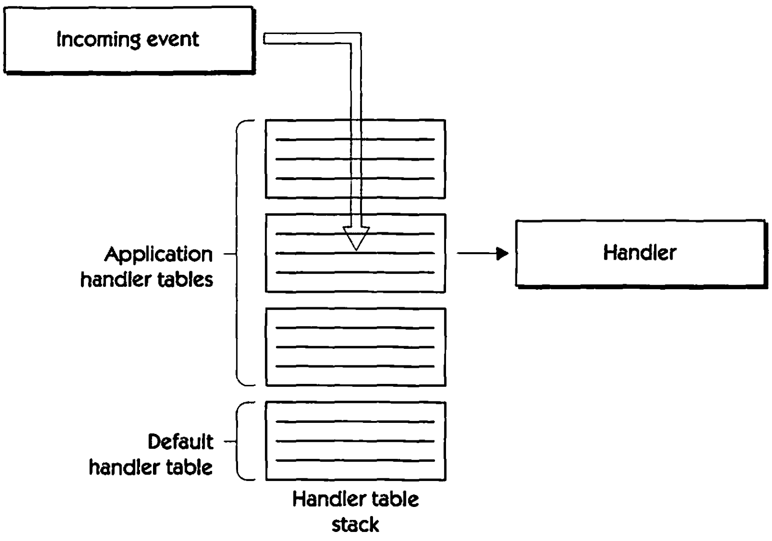
Dispatchers and Handler Tables

To dispatch Apple events within a process, the Apple Event Manager uses one or more Apple event dispatchers. An **Apple event dispatcher** consists of a queue of incoming events and a stack of handler tables. The Apple Event Manager looks through a dispatcher's handler tables to find handlers for events arriving in the queue. Mac OS 8 provides a default Apple event dispatcher for every cooperative program, and any program may create additional dispatchers as necessary.

A **handler table stack** consists of one or more handler tables. A handler table stack for a cooperative program always contains a default handler table and, usually, one or more application handler tables installed by a program. A **default handler table** contains default handlers installed by the operating system. The default handlers interpret standard events (such as Mouse Down when the user presses the mouse button) and, if necessary, route them to the appropriate panels in a window.

A developer uses the Apple Event Manager to install Apple event handlers in one or more **application handler tables**, which a program can add to or

FIGURE 14.1 An Apple event dispatcher calling a handler from a cooperative program



remove from the handler table stack at any time. Most developers add their own handler tables to implement the unique behaviors of their applications. For example, when a user double-clicks a document icon in the Finder, the Finder sends the Open Document event to the application that created the document. The application responds by performing its own operations for opening the document, such as loading it or mapping it into memory. Therefore, every Mac OS 8 application that opens documents provides its own handler for the Open Document event.

When the Apple Event Manager searches for an event's handler, it starts from the top of the stack and looks down the stack of handler tables until it finds a match. The default handler table for a cooperative program always resides at the bottom of a handler table stack. By stacking one handler table on top of another, a developer can augment or override the behavior defined by the lower table with the behavior defined by the higher table.

Figure 14.1 illustrates how the Apple Event Manager uses an Apple event dispatcher to dispatch an event. When an event arrives, the Apple Event Manager wakes up the task that called AEReceive and searches the stack of handler tables for a matching handler.

A developer can create two kinds of Apple event handler tables:

- **Unfiltered handler table.** When an unfiltered handler table contains no handler for a particular event, the Apple Event Manager passes the event to the next handler table in the stack.
- **Filtered handler table.** If a filtered handler table contains no handler for a particular event, the Apple Event Manager immediately suspends the event, which remains in the event queue. When the filtered table is removed from the handler table stack, the Apple Event Manager passes any suspended events to the next handler table, in the order in which they were received.

Unfiltered tables are appropriate for most situations. Filtered handler tables are useful when an application is in a modal state. For example, an application presents an alert box to warn the user or to report an error. An alert box panel places the application in a modal state requiring the user to click a button to acknowledge or rectify the problem. To suspend all events directed at the application except button clicks, an alert box panel automatically installs a filtered handler table in the handler table stack for the application's default dispatcher. After the user clicks a button to dismiss the alert box, the alert box panel removes the filtered handler table. Any events suspended during the display of the alert box (for example, a disk-inserted event) are then dispatched to the appropriate handlers in the order in which they were received. Developers can also install filtered handler tables directly to control modal states.

In the OpenDoc environment for Mac OS 8, every part editor has its own Apple event dispatcher, which supplies the part with the default handler table used by cooperative programs. A part editor can add handler tables to its handler table stack or remove them at any time. When OpenDoc sends an Apple event to a part editor, the part editor sends the event on to its Apple event dispatcher. The Apple Event Manager then searches the part editor's handler table stack and invokes the appropriate handler.

COMPATIBILITY NOTES

Standard System 7 Apple Events

Standard Apple events previously defined by Apple—for example, the required suite of Apple events discussed in *Inside Macintosh: Interapplication Communication*—are still supported in Mac OS 8 and play the same roles as they do in System 7.

Handlers

If an application doesn't need to handle a particular event, a developer doesn't need to install a handler for it. For example, a developer who uses the default text-handling supplied by editable text panels may not need to install handlers for text events.

If an application needs more information about an event before deciding whether to handle it, the application must install a handler for that event. For example, an application that supports the Get Data event to return certain kinds of data must install a handler that receives all Get Data events, including some the application may not be able to handle.

When the Apple Event Manager finds an entry for an event in a handler table, it passes the event to that handler. In most cases the handler simply responds to the event and returns a result code indicating it did so successfully. All processing of the event then ceases. If the handler can't handle the event for any reason or handles only part of it, the handler returns a result code that instructs the Apple Event Manager what to do with the event. For example, the handler can return a result code instructing the Apple Event Manager to continue its search through the handler table stack for a handler. Passing on the event in this way can also be useful if the handler performs preliminary handling only and a developer wants to take advantage of additional handling provided by handlers in lower tables.

If a handler understands the event but the event is impossible to handle—for example, a Get Data event directs a program to return the fifth paragraph of a document that only has four paragraphs—the handler should return an appropriate result to prevent the Apple Event Manager from continuing to search the handler table stack.

COMPATIBILITY NOTES

System 7 Apple Event Manager

Mac OS 8 supports all functions and data types defined by the Apple Event Manager in System 7. However, the Mac OS 8 Apple Event Manager replaces System 7 functions used to dispatch, receive, and send events with entirely new functions. For example, the AEReceive function in Mac OS 8 replaces the WaitNextEvent and AEProcessAppleEvent functions from System 7. To fully adopt the Mac OS 8 event model, developers must use the Mac OS 8 replacement functions.

Apple Event Manager functions used in System 7 to add data to or get data from Apple events are fully supported in Mac OS 8 and, in general, the use of these functions remains unchanged. The developer of a Mac OS 8 program can use most of these as described in *Inside Macintosh: Interapplication Communication*. However, Mac OS 8 applications must use new Apple Event Manager accessor functions for obtaining and clearing the data in descriptor records.

SCRIPTABILITY

Once a developer has adopted the Mac OS 8 event model in an application, it's very simple to make that application scriptable. The same Apple events that communicate user and system events can be used to automate scriptable applications. A scriptable application could allow scripts or other applications to send Apple events identical to those sent by the operating system. The operating system, for example, sends Apple events informing an application of commands that the user chose from its menus. The application's Apple event handlers respond to these Apple events by performing the commands. A sequence of command selections could be automated in a script that sends these same Apple events to the application.

A **script** is a series of statements, written in a scripting language, instructing a computer to perform various operations. A **Scripting language** is designed to automate and control programs and to be easier to learn and use than complex languages like C.

Suppose, for example, that a communications program and a separate page-layout program are both scriptable. A systems integrator or a technically adept user could write a script that instructs the communication program to dial an online information service at the same time every morning and download news articles relating to an important topic. Using the delegation capabilities of the Assistance Services described in Chapter 13, the script could then direct the page-layout program to launch, copy this information into a report template, and print the report. A user would then have an automatically prepared briefing first thing in the morning.

APPLE EVENTS AND THE HUMAN INTERFACE TOOLBOX

As you read in Chapter 12, human interface objects automatically perform much of their own event handling. For example, a push button become highlighted automatically as the user presses the mouse button and moves the cursor over the button. The Human Interface Toolbox supports this automatic behavior by providing the default Apple event dispatcher with default handlers for many standard events, including those involving the mouse, windows, and text input (whether entered by keystroke, voice, or other method). The Toolbox-supplied handlers automatically route most of these events to the targeted human interface objects, which provide methods that handle the specific events.

For example, Toolbox-supplied handlers typically determine the target window for an event and forward it to that window. Every window, in turn, is associated with an Apple event dispatcher that initially processes all events directed to the window. For further event processing, the handlers associated with the window dispatcher typically call the methods of one or more human interface objects displayed in the window.

By default, the target objects respond automatically to certain user actions that change an object's state—for instance, by zooming a window to the appropriate monitor, highlighting menu items as the user drags the cursor through them, changing the highlighting of radio buttons, and so on.

An application can associate application-specific behavior with a particular object in three principal ways:

- After an application instantiates an object, the application can install an application-defined function that the Toolbox calls whenever there's a state change. The application-defined function then performs application-specific behavior.
- An application can install its own handler tables with the default Apple event dispatcher. The handlers in these tables are called before those in the default handler table.
- An application can subclass the human interface objects classes and override selected methods. The overridden methods perform application-specific behavior.

The ability of the standard human interface objects to respond to user interaction appropriately is not limited to low-level events such as mouse events and keyboard events. Mac OS 8 supports higher-level events that correspond to interface abstractions such as "select object." The default Toolbox event handlers translate mouse events, keyboard events, speech events, and other events generated by input devices into higher-level events that an application handles the same way regardless of the type of input device.

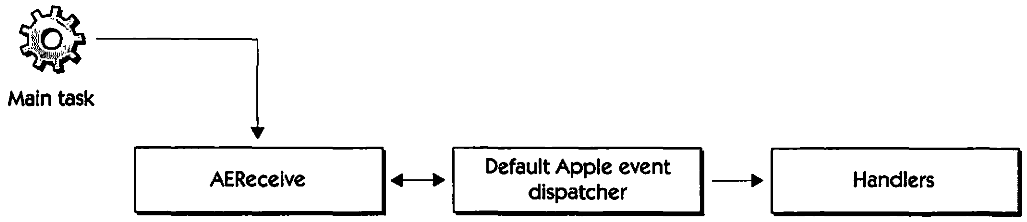
EVENT HANDLING FOR ONE OR MORE TASKS

When any task calls AEReceive, the task specifies the Apple event dispatcher in which it's interested. Only the main task of a cooperative program can specify the default Apple event dispatcher. Other tasks using AEReceive specify developer-supplied Apple event dispatchers. A developer can associate tasks with Apple event dispatchers as follows:

- one task with one dispatcher
- multiple tasks with multiple dispatchers
- multiple tasks with one dispatcher

Identifying the particular arrangement of tasks and dispatchers appropriate for a program is a design decision. The first two models are appropriate for both cooperative programs and server programs. The third model, which associates multiple tasks with a single dispatcher, is intended for use by server programs only. All three models may be combined in various ways to support

FIGURE 14.2 One task, one dispatcher for a cooperative program



more complex relationships among tasks and dispatchers and to assign operations to server programs.

One Task with One Dispatcher

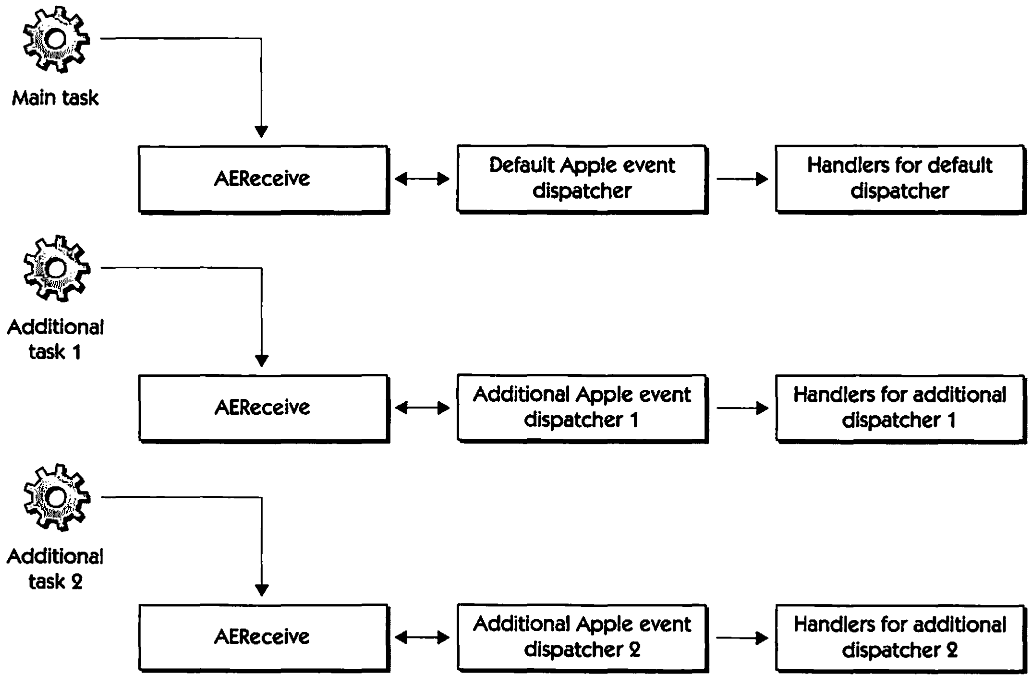
Figure 14.2 shows the simplest case: a main task associated with the default Apple event dispatcher associated with a cooperative program. In this arrangement, a single task is responsible for all event handling. The AEReceive function blocks the main task until an event arrives for the task. The Apple Event Manager then invokes the main task and calls the Apple event handler designed to respond to the event.

Multiple Tasks with Multiple Dispatchers

A developer must associate the main task of a cooperative program with a single Apple event dispatcher—the default Apple event dispatcher, as in Figure 14.2. A developer can create additional dispatchers for tasks that use reentrant services only. Figure 14.3 illustrates this arrangement. Another alternative, described in the following section, is to associate additional tasks with a single dispatcher.

A developer can implement the model shown in Figure 14.3 in several ways. It's possible, for example, to route events to a particular dispatcher. All human interface-related events must be routed through the default Apple event dispatcher, but a developer can design a program to route other events to other dispatchers. For example, suppose a graphics program has a menu command that transforms an image in some way by performing a series of calculations. The handler invoked by that command can, in turn, send an Apple event to a different dispatcher associated with a separate task. This separate task then performs the calculations. The main task is then free to continue responding to the user while the second task, which doesn't involve the human interface, executes in the background.

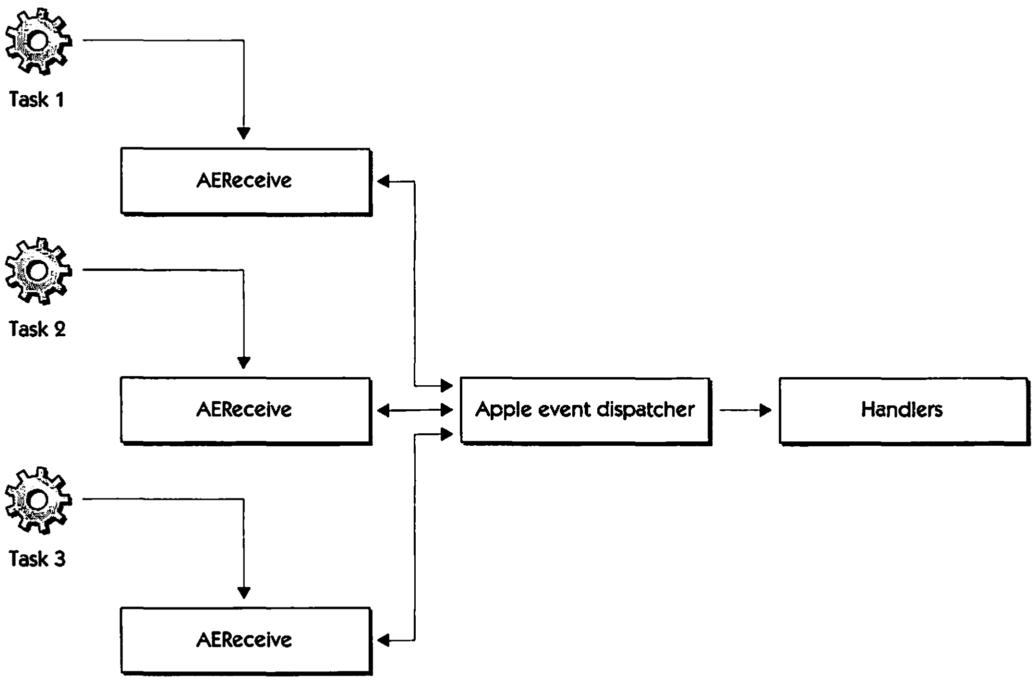
FIGURE 14.3 Multiple tasks, multiple dispatchers


Main task
AEReceive
Default Apple event dispatcher
Handlers for default dispatcher
Additional task 1
AEReceive
Additional Apple event dispatcher 1
Handlers for additional dispatcher 1
Additional task 2
AEReceive
Additional Apple event dispatcher 2
Handlers for additional dispatcher 2

When the second task needs to inform the user of its progress, the handler that's performing the calculation can direct the default Apple event dispatcher to update a progress indicator. When the handler has completed its calculations, it can direct the default dispatcher to invoke the handler that actually draws the transformed image.

Because Mac OS 8 permits a program to use multiple tasks in addition to its main task, the graphics application in this example could actually perform transformation calculations on several different images, starting each calculation at a different time and performing them all concurrently. Thus, the main task could be drawing the results of one calculation to the screen while another task is in the middle of calculating a transformation for a second image. At the same time, another task can begin calculating a transformation for a third image. In effect, this kind of arrangement allows a developer to create a "server within the application," even though the additional tasks don't necessarily have to be implemented as independent server programs.

FIGURE 14.4 Multiple tasks, one dispatcher



Multiple Tasks with One Dispatcher

Figure 14.4 shows multiple tasks calling AEReceive and specifying the same Apple event dispatcher. Each task has its own entry point and begins executing at a different time. The tasks don't necessarily have to be identical, but they must use the same handler table stack and must be equally capable of dealing with incoming events. Because tasks are preemptively scheduled for execution, their access to data must be synchronized. Therefore, a developer must make all handlers fully reentrant in this arrangement.

Because only one task in a cooperative program may use handlers that access the cooperative services, no additional tasks for that program can share the main task's Apple event dispatcher. Therefore, this arrangement is useful only for server programs. For example, a database program that receives requests continuously from several sources can spawn a series of identical tasks associated with the same Apple event dispatcher. All of these tasks share the same handler table stack. The Apple event dispatcher pairs each task with each incoming request and looks up the corresponding handler in the handler table stack. Thus, the database can handle a series of requests concurrently.

SUMMARY

The Mac OS 8 event model is based on Apple events. Events involving the human interface—for instance, user actions with the mouse and keyboard—are automatically packaged by the operating system as Apple events. Tasks that need to communicate with one another can also package their messages as Apple events.

When the main task of a cooperative program uses AEReceive, it yields execution eligibility whenever there are no events for it to respond to. This allows the Process Manager to synchronize access to the cooperative services, and in this way, the program participates in the operating system's cooperative scheduling policy. (Backward compatibility support also allows older applications using the Event Manager from System 7 to participate in cooperative scheduling.) Tasks created for server programs and additional tasks spawned by cooperative programs can also use AEReceive. These tasks are preemptively scheduled by the operating system.

When there are no events for a task, AEReceive blocks it. When there is an event, the task becomes eligible for execution. As the task begins to execute, AEReceive dispatches the event to an Apple event handler designed to respond to that event. After its handler executes a response to the event, AEReceive blocks the task again until another Apple event arrives.

Apple event handlers for a task are maintained in a stack of handler tables. For every cooperative program, a default handler table contains handlers that respond to many standard events, particularly those involving the human interface. The default handler table always resides at the bottom of a handler table stack. To override the handlers in the default handler table and to supply handlers for program-specific behavior, a program can dynamically add application handler tables to its handler table stack and dynamically remove them.

A handler table stack and a queue of incoming events constitute an Apple event dispatcher. A program with one task uses a single dispatcher to route events to the appropriate handlers within the program's handler table stack. A program with multiple tasks can support the Mac OS 8 event model by following one of these event-handling approaches:

- ▶ Assign a different Apple event dispatcher to each task.
- ▶ Use a single Apple event dispatcher to respond concurrently to all events directed at any of the program's tasks.

The first approach is especially useful for a cooperative program that uses the main task to handle human interface events and other tasks to perform background processing. The second approach requires all of a program's handlers to be fully reentrant. This approach is appropriate only for server programs, which use only reentrant operating system services.

PLANNING A PRODUCT FOR MAC OS 8

If you're a developer, the best way to begin preparing a product that takes advantage of the Mac OS 8 event model is to separate the code that controls an application's user interface from the code that responds to the user's manipulation of the interface. This is called **factoring** an application. A fully factored application, as described in *Inside Macintosh: Interapplication Communications*, translates user actions into Apple events that the application sends to itself. Factoring not only supports the Mac OS 8 event model but also allows applications to be controlled by any scripting language, such as AppleScript, that's based on the Open Scripting Architecture (OSA).

15

Landmark Imaging and Multimedia Technologies

Since its inception, the Macintosh Operating System has featured innovative technologies that have drawn users and developers to its platform. Bit-mapped graphics, desktop publishing, and built-in networking were early features of the operating system. System 7.5 incorporated support for multimedia publishing and playback, video conferencing, Internet connectivity, and three-dimensional graphics. Using these landmark technologies, developers have created astonishing applications for the Mac OS.

Mac OS 8 inherits these technologies, refines them, and integrates them into the operating system. This chapter describes the imaging and multimedia technologies carried forward from System 7.5 and explains how they take advantage of Mac OS 8 capabilities. Chapter 16 describes the networking features inherited from System 7.5.

Mac OS 8 implements the imaging and multimedia services from System 7.5 as memory-efficient shared libraries. Compared with System 7 applications, applications using these technologies in Mac OS 8 share the CPU more efficiently because of the operating system's priority-based multitasking capabilities. Also, the virtual memory system automatically helps these applications make more efficient use of physical memory. The concurrent I/O system introduced in Mac OS 8 further improves the performance of imaging and multimedia-based applications. For example, video data can be read from disk faster in Mac OS 8 than in System 7.5, and pictorial images can be shared across a network faster.

KEY TERMS AND CONCEPTS

- **Multimedia** refers to combining multiple forms of communication to facilitate the transmission of ideas and information. Multimedia applications supported by Mac OS 8 allow a user to combine text, pictures, video, sounds, music, and other types of data into multimedia documents.
- **Imaging** refers to the construction and display of graphical information. Graphical information can consist of shapes, pictures, and text and can be rendered on devices such as screens and printers. All graphical portions of a multimedia document, for example, are processed and displayed in Mac OS 8 through the use of imaging services available as part of the operating system.
- **QuickDraw 3D** is a cross-platform, interactive 3D graphics technology.
- **ColorSync** is an industry-standard architecture underlying the operating system's color-matching services. These services allow users to move color images reliably from one device to another (such as from a scanner to a video display and then to a printer) and from one operating system to another.
- **QuickDraw GX** is a collection of graphics, typography, and printing services that provide applications with sophisticated color publishing capabilities.
- Introduced with the first version of the Macintosh Operating System, **QuickDraw** performs onscreen graphics operations on behalf of applications and the operating system. A precursor to the more sophisticated capabilities of QuickDraw GX, QuickDraw remains a fully supported graphics system in Mac OS 8.
- **QuickTime** is a collection of cross-platform operating system services that allow applications to control time-based data, such as video and music.
- **QuickTime VR** is a cross-platform service offering two kinds of virtual reality experiences: a panoramic experience enabling users to explore 360-degree scenes and an interaction experience allowing users to "pick up" and interact with objects.
- **QuickTime Conferencing** is a cross-platform collaboration and communications technology that allows users to broadcast and view real-time digital audio, text, images, and video.

MAJOR POINTS OF INTEREST

When the Macintosh computer was introduced in 1984, its most striking characteristic was its well-designed graphical user interface. The interface featured onscreen objects—such as icons, menus, and text displayed in multiple fonts, sizes, and typestyles—that users could manipulate directly. Users could control the computer by thinking visually and interacting with it visually. This visual interaction helped make the computer easier and more fun to use.

With the advent of the PostScript page description language and the LaserWriter printer, the imaging strengths of the Macintosh computer were quickly applied to a new use: desktop publishing. With a Macintosh computer and a LaserWriter printer, casual users could prepare visually appealing printed material without resorting to costly professional publishing services.

Apple has continued to advance the human interface of its operating system by supporting other types of media in addition to graphics and text. System 7 users can manipulate video, animation, three-dimensional objects, voice and sound, MIDI music, and interactive panoramic scenes as easily as text and two-dimensional shapes. Whereas System 7.5 supports these forms of data with optionally installed system extensions, Mac OS 8 fully incorporates the technologies that allow users to create and communicate with these forms of data.

A **system extension** is a file in System 7 containing code that's loaded into memory at system startup time.

QuickDraw for black-and-white screens and the Printing Manager for dot-matrix printers constituted the first imaging systems available on the Macintosh computer. When the color version of QuickDraw was introduced for Macintosh computers with color displays, users could work with photo-realistic images on screen. Since then, the imaging capabilities of the operating system have been greatly enhanced by ColorSync, QuickDraw GX, and QuickDraw 3D.

Introducing operating-system support for video, music, and other time-based data in System 7, QuickTime laid the foundation for multimedia capabilities in the Mac OS. QuickTime VR, QuickTime Conferencing, QuickTime Music Architecture, and other QuickTime-based technologies have further extended the multimedia capabilities of the Mac OS.

If you haven't already done so, read "How to Navigate the Book CD-ROM" in the preface for information about using the CD-ROM version of this book. In the CD-ROM version of the book, the screen shots presented in this chapter are animated.

TAKING ADVANTAGE OF MAC OS 8 CAPABILITIES

Users fond of the System 7.5 imaging and multimedia technologies will appreciate the extra performance and stability that the Mac OS 8 platform affords. The next several sections describe why these technologies perform better in Mac OS 8 than in System 7.5.

Full-Time Availability

Most of the advanced services are available in System 7.5 only after the user installs them as system extensions. System extensions take up memory from the time the user turns on the computer until the user turns the computer off—whether or not any application uses them. Because of these memory demands, very few System 7.5 users have all of the imaging and multimedia services running simultaneously on their computers. To use one service, such as QuickDraw GX, a user might have to turn off another service, such as QuickTime, and restart the computer—a noticeable inconvenience.

The lack of full-time availability of these services in System 7.5 also causes extra work for developers. System 7 developers often need to write multiple code paths based on the availability of these services. For example, a System 7 developer might need to write several code paths for drawing images in an application: one path for systems using QuickDraw GX and another path for systems that aren't; one path for systems using QuickDraw 3D and another path for systems that aren't; and perhaps additional paths based on the availability of ColorSync.

By contrast, Mac OS 8 makes its imaging and multimedia services available at all times on every system. Users and developers can rely on the presence of these technologies on every Mac OS 8 system. These technologies can be made available full-time because they take advantage of Mac OS 8 memory efficiency features.

Making Efficient Use of Memory in Mac OS 8

In Mac OS 8, all imaging and multimedia services are implemented as shared libraries. Recall that the dynamic, execution-time preparation of these shared libraries reduces memory requirements by allocating memory for a shared library only when it's referenced by a program or another library. The shared library for QuickTime, for example, is mapped into memory only when an application references it at launch time. All subsequently launched programs using QuickTime share the same memory-mapped copy of QuickTime code. When the last application to use this library quits, the operating system releases the memory areas allocated to QuickTime code.

Shared libraries are discussed in **Chapter 8,** and the virtual memory system in **Chapter 6.**

Imaging and multimedia-based applications tend to require large amounts of memory for their own code and for the documents they create. However, the efficient virtual memory system of Mac OS 8 allows the user to work with many more large applications and documents than can fit in the physical memory of a computer running Mac OS 8—or even in the virtual memory of a computer running System 7.5.

The Dynamic Storage-Allocation Service is described in **Chapter 7.**

With the exception of QuickDraw, the imaging and multimedia services in Mac OS 8 use the reentrant, pointer-based Dynamic Storage-Allocation Service to allocate and release temporary data storage. For this reason, imaging and multimedia services perform much better than their System 7.5 ancestors.

In System 7.5, some of these services use the nonreentrant, handle-based Memory Manager, while others implement their own private memory models, adding to the code overhead. For several reasons, including its reliance on global data shared simultaneously by cooperative programs, QuickDraw in Mac OS 8 continues to use the Memory Manager—a cooperative service supplied for System 7 application compatibility.

Taking Advantage of Mac OS 8 Multitasking Capabilities

The multitasking capabilities of Mac OS 8 are described in **Chapter 4**, and the ways multithreaded programs take additional advantage of these capabilities are described in **Chapter 5**.

The multitasking capabilities of Mac OS 8, in combination with its concurrent I/O system, offer more efficient data processing to increase the performance of all products, but especially multimedia products. For instance, a video-editing program can read data from a CD-ROM disk in one thread of execution, write data to a hard disk in another thread of execution, and yet remain highly responsive to the user in a third thread. Even while data is coming from the CD-ROM or going to the hard disk, the program can keep the CPU busy executing other operations.

Through its priority-based scheduling rules, the Mac OS 8 microkernel supports the real-time processing needs of multimedia-based applications. By placing multimedia operations in a separate real-time task, a multithreaded application can perform time-critical multimedia operations without interruption. For example, a multimedia authoring application can use a task outside of its main task to capture and process video data in real time or to play uninterrupted sound.

Most graphics and multimedia operations that involve drawing to the screen are implemented as cooperative services in Mac OS 8. However, QuickDraw GX is implemented as a reentrant service. Multithreaded programs can take advantage of preemptive scheduling by placing image-processing operations in background tasks. A multithreaded graphics program, for example, can use its main task to call QuickDraw GX to perform screen drawing operations. The program can also use QuickDraw GX in an additional background task to preprocesses data that the program's preparing to draw onscreen. Such background processing might, for instance, entail measuring the text and calculating the line breaks of a large document. When developers use QuickDraw GX to draw to the screen, they do so exclusively from the main tasks of their cooperative programs. Confining all screen-drawing operations to the cooperative scheduling environment ensures that graphical operations involving user interaction are properly serialized.

Using the Concurrent I/O System

The I/O system is discussed in **Chapter 11**.

The concurrent I/O system introduced in Mac OS 8 further improves the performance of imaging and multimedia-based applications. Compared with System 7.5, for example, more 3D images can be shared across a network in a given amount of time.

Taking Advantage of System Stability

As you've read in previous chapters, Mac OS 8 provides a highly stable platform. Server programs, such as the font scalers used by the operating system, operate on data in memory areas that are protected from errors in application code. Data used by all critical portions of the operating system, such as the I/O system and the microkernel, are fully protected from corruption by application-level software. And because all code is mapped into read-only memory areas, code can never be corrupted while being executed.

Although Mac OS 8 offers these safeguards for system stability, one cooperative program can potentially corrupt the data of another cooperative program in their shared address space. However, developers can design their applications to further lessen any vulnerability to errors in cooperative programs. For instance, a multimedia application can use a server program to reliably perform long video-processing operations or to dependably transmit music to remote users across the World Wide Web.

INTEGRATED IMAGING SYSTEMS

In System 7.5, a developer can choose from among several graphics models when writing an application:

- QuickDraw for drawing onscreen graphics and text
- the Printing Manager for printing documents
- QuickDraw GX for printing documents and drawing onscreen graphics and text
- QuickDraw 3D for drawing three-dimensional shapes
- WorldScript I and II for managing text and fonts for international markets
- ColorSync for ensuring accurate color matching among devices, such as scanners, video displays, and printers.

In System 7.5, QuickDraw GX, QuickDraw 3D, WorldScript I and II, and ColorSync are available to users as optionally installed system extensions. System 7 developers can always depend on the built-in support of QuickDraw, but they can't rely on the presence of these other imaging systems on users' computers.

Developers can, however, rely on the presence of these technologies on every Mac OS 8 system. This integration of imaging services offers developers a single, ever-available set of programming interfaces. The full-time availability of imaging services eliminates the need for redundant code paths in an application and allows developers to take full advantage of these services. The powerful QuickDraw GX line layout technology, for example, is always available to every Mac OS 8 application.

COMPATIBILITY NOTES

The System 7.5 Imaging Managers

The Mac OS 8 imaging architecture supports all of the programming interfaces defined by QuickDraw, QuickDraw GX, QuickDraw 3D, ColorSync, and WorldScript I and II. Apple intends to support these graphics models beyond Mac OS 8 so that developers' code based on these technologies will continue to run in the future.

Mac OS 8 also supports System 7 applications using the Printing Manager programming interface. In Mac OS 8, this programming interface translates Printing Manager function calls to QuickDraw GX function calls. In Mac OS 8, therefore, all printing is based on the QuickDraw GX print model. To get the best performance on Mac OS 8 and to ensure that their print-handling code continues to work on future versions of the Mac OS, developers should use the QuickDraw GX programming interface for printing instead of the Printing Manager.

QuickDraw 3D

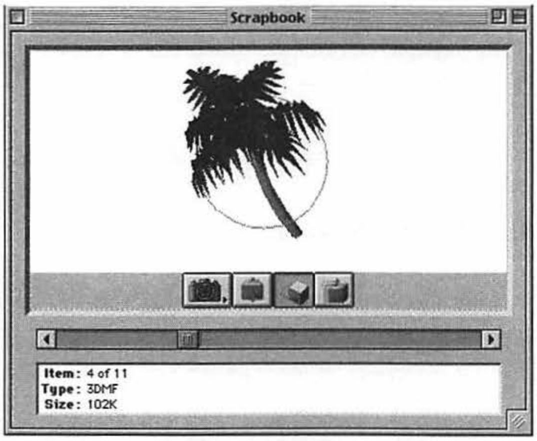
QuickDraw 3D supplies a graphics library that allows Mac OS 8 developers to define three-dimensional (3D) models, apply colors and other attributes to parts of the models, and create images of those models. Developers have already used this library to incorporate interactive 3D modeling, simulation and animation, data visualization, and computer-aided drafting and design (CAD/CAM) into their System 7.5 applications.

QuickDraw 3D serves a wide range of users. By eliminating many of the obstacles of previous 3D graphics technologies, QuickDraw 3D moves 3D graphics from the domain of highly specialized artists and equipment to that of typical users and personal computers. Users can create shapes in 3D using a variety of input devices, such as 3D track balls, pressure-sensitive tablets, and 3D scanners.

Distributed with the operating system, the **Scrapbook** is a program that lets users store text, graphics, sounds, movies, 3D objects, and other frequently used information.

Using an application developed with QuickDraw 3D, users can work with the actual 3D graphics objects rather than the less detailed wire-frame representations commonly associated with 3D graphics applications. Whether pasting editable 3D graphics into documents and presentations or directly creating and modifying 3D art, users work with these 3D objects as easily as with standard two-dimensional graphics. For example, Figure 15.1 shows a QuickDraw 3D object saved in the Scrapbook. As with any data saved in the Scrapbook, the user can drag-and-drop this object into open documents.

FIGURE 15.1 3D data saved in the Scrapbook for placement in standard documents



Item: 4 of 11
Type: 3DMF
Size: 102K

Within the Scrapbook, the user can move this object around three axes of rotation, as the CD-ROM animated version of this figure demonstrates. The user can also zoom in to take a closer view of the object or zoom away.

The QuickDraw 3D human interface, by which users access and control the tools in a QuickDraw 3D-based application, applies consistent behavior to the application tools, making these elements an extension of the overall Mac OS human interface. Using the QuickDraw 3D Human Interface Toolkit, developers avoid reinventing common interface elements, and users can apply their 2D interface knowledge to 3D applications.

Incorporating QuickDraw 3D into Applications

QuickDraw 3D is designed to be useful to a wide range of developers, from those with little knowledge of 3D modeling concepts and rendering techniques to those with extensive experience in these areas. Developers can use QuickDraw 3D's capabilities as much or as little as they need to.

- ▶ Developers can use QuickDraw 3D's industry-standard file format and file-access functions simply to read and display 3D graphics created by other applications. For example, a word-processing application might allow users to import pictures created by 3D modeling applications.
- ▶ A developer who chooses not to learn the core QuickDraw 3D application programming interface can use the 3D Viewer mechanism to display 3D objects in a window and allow users limited interaction with those objects.

- Using QuickDraw 3D's full capabilities, developers can create applications that perform interactive 3D modeling and rendering, animation, data visualization, or any kind of sophisticated 3D-data interpretation and display.

QuickDraw 3D includes a complete library of predefined 3D objects. A QuickDraw 3D **object** is a shape that contains or references specifications about how and where the object should be drawn, and this information travels with the object. Developers can create multiple instances of any type of 3D object and assign them individual characteristics. Users can change an object's appearance by rotating it, scaling it, or otherwise transforming it. Although QuickDraw 3D provides a large set of 3D objects and operations, it's also designed for easy extensibility so that developers can add custom capabilities to their objects.

QuickDraw 3D also supports standard lighting types and illumination algorithms. Several types of lights may be cast on an object, and these lights can illuminate objects and scenes in different ways, according to various levels of quality and computational complexity.

3D Metafile Format

One of the barriers to the widespread use of 3D has been the inability to use 3D images outside the program that created them. QuickDraw 3D solves this problem by reading and writing data in its cross-platform 3DMF (3D Metafile) format. This format allows 3D objects created by one application to be used by other applications. The 3DMF format specifies objects, their properties, and properties of the scene that contains them (including orientation, lighting, camera, and shading). The file format preserves all object properties in either text or binary formats.

3DMF also allows applications to save and exchange unique attributes. For example, if a program uses a custom effect to create a 3D object, it would normally be impossible to view that object in any other program without losing that effect. Saved as a 3DMF file, however, the object retains its unique, application-specific attributes in other programs—such as web browsers—even if those programs don't support this feature directly. Because 3DMF technology provides consistent capabilities and performance across the Mac OS, Windows, and UNIX operating systems, 3DMF provides a standard and convenient way for users to exchange 3D graphics across the Internet.

The 3D Viewer

The 3D Viewer provides a simple mechanism for users to view and interact with 3D objects. A control strip at the bottom of the 3D Viewer window contains tools—a camera angle button, a distance button, a rotate button, and a zoom button—that let users manipulate the location and orientation of their point of view. Even if they don't want to learn the core QuickDraw 3D application programming interface, developers can easily incorporate the 3D Viewer into their applications, giving them the ability to display 3D data.

Added Acceleration Capabilities

Acceleration capabilities built into QuickDraw 3D provide superior performance for computers that contain accelerator cards. For example, Apple Computer's QuickDraw 3D Accelerator Card makes QuickDraw 3D run up to 12 times faster. With this increased performance, QuickDraw 3D can offer additional capabilities, such as Gouraud shading (by which users improve an object's color), texture mapping (by which users apply a picture to a surface), and anti-aliasing (which smooths the edges of 3D objects to prevent jagged images). With these accelerator cards, QuickDraw 3D can produce more visually complex 3D objects without increasing the complexity of the object itself.

By supporting Constructive Solid Geometry (CSG), QuickDraw 3D acceleration also lets applications control the interaction of two or more objects. For example, an application might punch holes in objects or fuse several objects together.

Developers of games or other 3D-intensive software can use **QuickDraw 3D RAVE** (Rendering Acceleration Virtual Engine), the foundation technology used in QuickDraw 3D. RAVE is an optimized hardware abstraction layer that allows programmers to code directly to 3D graphics accelerator cards for maximum performance.

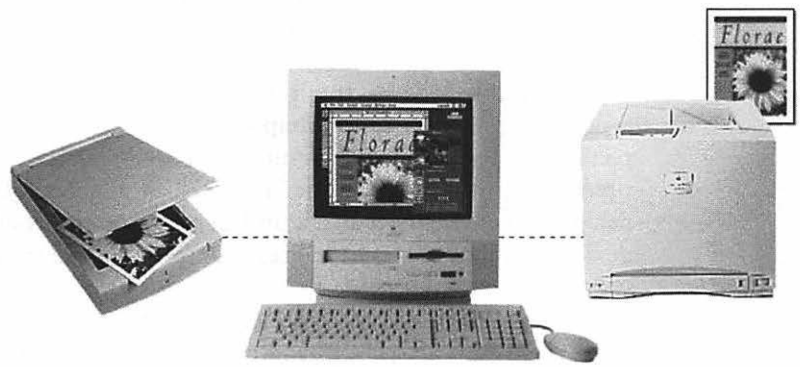
ColorSync

ColorSync is the color-matching architecture for the Mac OS. It allows users to get more predictable and accurate color from their applications, scanners, digital cameras, displays, and printers. For example, ColorSync ensures that the colors appearing in a scanned photograph of a daisy, as in Figure 15.2, match the colors of that image when it's displayed on a video screen and printed on a printer.

The components of the **RGB color space** are the red, green, and blue intensities that make up a given color. RGB color spaces are used mainly for displays and scanners. Based on the colors cyan, magenta, yellow, and black, the **CMYK color space** models the application of inks and dyes to paper.

Before ColorSync was available, computer users, especially publishing professionals, faced the problem of how to display the same image on more than one device without changing the colors perceptibly. Different imaging devices such as scanners, displays, and printers work in different color spaces. **Color spaces** are models, such as RGB and CMYK, that specify how color information is represented. Even using the same color space, different devices can have different ranges of color, called **gamuts**. A gamut measures the range of the lightness, darkness, and density of colors in a given color space. For example, color video displays from different manufacturers all use the RGB color space, but they have different RGB gamuts. Printers that work in CMYK space vary drastically in their gamuts, especially if they use different printing technologies. Even a single printer's gamut can vary significantly with the ink or the

FIGURE 15.2 Color matching across devices



type of paper it uses. It's easy to see that conversion from RGB colors on an individual video display to CMYK colors on an individual printer can lead to unpredictable results.

To address these problems, ColorSync provides color-matching services. ColorSync can automatically convert colors from one color space to another and adjust (or "match") these converted colors from the gamut of one color space to that of the other. For even finer color control, an application using ColorSync can allow the user to perform quick and inexpensive color proofing, see in advance what colors cannot be printed on a printer, and adjust the colors so that images rendered on different devices match exactly.

To provide its color-matching services, the ColorSync Manager uses one or more color management modules (CMMs) and profiles. A **CMM** implements color-matching and gamut-checking services. A **profile** provides a means of defining the color characteristics of a given imaging device. The ColorSync Manager programming interface allow applications and device drivers to provide ColorSync support, create and manage profiles, and create CMMs that respond to requests from applications and device drivers.

When an application uses ColorSync directly to match colors between devices, it must specify the profile for each device when calling a ColorSync color-matching function. As experienced QuickDraw GX developers are aware, however, an application using QuickDraw GX doesn't need to use the ColorSync programming interface directly. On behalf of applications, QuickDraw GX automatically uses ColorSync to perform color matching across devices.

QuickDraw GX

QuickDraw GX is an imaging architecture that supports powerful graphics, typography, and printing capabilities in the Mac OS. These capabilities are especially suited to typographical and color publishing needs, from mainstream business communications to high-end commercial publishing.

QuickDraw GX has built-in support for multiple types of graphic and typographic images called shapes. Examples of shapes include lines, points, rectangles, polygons, curves, multiple-curve paths, simple text, text with multiple styles, sophisticated line layouts, and pictures. Developers define shapes using measurements that are independent of the resolution of any imaging device, and developers have great control over the stylistic variations of these shapes.

A shape encapsulates specifications about how and where it should be drawn. All of a shape's attributes—for instance, size, color, fill, perspective, and line thickness—are stored with that shape. Developers don't have direct access to the internal data structures of a shape, but instead make function calls to examine or modify the values of a shape's properties. This object-oriented approach to encapsulating data within shapes greatly simplifies the developer's tasks.

QuickDraw GX allows developers to perform sophisticated geometric operations on geometric shapes. These operations include, editing, measuring, simplifying, and converting from one type of shape to another. Developers can transform shapes by relocating, scaling, skewing, adding perspective to, and otherwise distorting them.

QuickDraw GX uses device-independent colors and offers built-in support for ColorSync so that colors in a document created by an application using QuickDraw GX translate reliably to different video displays and printers. QuickDraw GX also provides built-in support for multiple color spaces including luminance (for grayscale), RGB (for display screens), YIQ (for color video broadcast), CMYK (for printing), HSV and HLS (for user selection of colors in an application), and CIE (for colorimetrics).

In addition to these and other advanced graphics features, QuickDraw GX provides applications with sophisticated printing and typographic capabilities.

QuickDraw GX Printing

Mac OS 8 fully adopts the QuickDraw GX printing architecture. This architecture supplies an intuitive and flexible human interface for users. For developers, QuickDraw GX provides an easy way to offer application-specific printing capabilities, and it defines a simplified model for creating printer drivers.

A **printer driver** is a plug-in that controls how the contents of a document are spooled, rendered, and sent to a specific output device.

In QuickDraw GX, printers are represented by icons on the computer desktop. A user can print a document by dragging the document icon to a printer icon. The user can have multiple printer icons on the desktop and can print to them simultaneously. The user can redirect a print job from one printer to another. The user can also specify a separate page format for each page of a document. For example, suppose a document consists of three pages—a page of addresses, a business letter, and a spreadsheet. The printed version of the document can have three formats: an envelope format for the addresses, a portrait format for the business letter, and a landscape format for the spreadsheet. When closing this document, the user can save this printing setup information with the document.

Developers can provide additional print options to their applications, giving users additional flexibility over printing jobs. For example, an application might supply a printing extension allowing the user to print, in light gray, the word *Confidential* on every page of a document.

For printer driver developers, the QuickDraw GX print architecture streamlines driver development. Compared to the Printing Manager architecture, for example, QuickDraw GX substantially reduces the code required for printer drivers, sometimes by as much as 95 percent.

COMPATIBILITY NOTES

Printing Manager Printer Drivers

The QuickDraw GX printer driver model completely replaces the model used by the Printing Manager. Therefore, Printing Manager printer drivers do not work in Mac OS 8.

QuickDraw

Experienced Mac OS developers are very familiar with QuickDraw, the graphics architecture used to draw human interface elements on all previous versions of the Macintosh Operating System. The images that QuickDraw manipulates can consist of shapes, pictures, and text and can be displayed on such devices as screens and printers. Over the years, QuickDraw has evolved to accommodate the growing graphics capabilities of the Mac OS. Each new generation of QuickDraw has maintained compatibility with those that preceded it, while adding new capabilities and expanding the range of possible display devices. This evolutionary approach has helped to ensure that applications written for earlier Macintosh models continue to work as more powerful computers are developed.

The development of QuickDraw has progressed along these three main stages:

- Basic QuickDraw was designed for the earliest Macintosh models with their built-in black-and-white screens. System 7 added new capabilities to basic QuickDraw, including support of the offscreen graphics world—an environment for preparing complex images before displaying them on the screen. A subset of basic QuickDraw capabilities also supplies the basis for imaging on the Newton operating system for hand-held computers.

- The original version of Color QuickDraw was introduced with the first Macintosh II systems. This first generation of Color QuickDraw could support up to 256 colors.
- An updated version of Color QuickDraw (originally introduced as 32-Bit Color QuickDraw) was shipped as part of System 7. This version was expanded to support up to millions of colors.

QuickDraw is designed to create images on 72-dpi devices, such as video displays and older dot-matrix printers. Compared to QuickDraw, QuickDraw GX offers far more sophisticated graphics features, such as device-resolution independence, data encapsulation, automatic support for ColorSync, and advanced typographic capabilities. However, QuickDraw is used extensively by a great number of developers, and Apple intends to continue supporting it in future versions of the Mac OS.

Typography

Typography is the arrangement and appearance of printed characters. Desktop publishing has brought sophisticated typography into wider user than ever before. Mac OS 8 inherits the full legacy of Mac OS typographic features—including those previously available only with the optionally installed QuickDraw GX graphics system.

Fonts

A **font** is a complete set of glyphs in one typeface and style. A **glyph**, in turn, forms the graphical representation of a particular character. Since System 7, the Mac OS has used two types of fonts: bitmapped fonts and TrueType fonts. Bitmapped fonts were the only ones available on Macintosh computers until the introduction of System 7. In a bitmapped font, a glyph is an individual bitmap designed at a fixed point size and style for a particular display device. If the user requests a font that is not available in a particular size, QuickDraw can scale a bitmapped font to a different size to create the required glyphs. However, a scaled bitmap usually appears slightly jagged.

A **bitmap** is a data structure that represents the positions and states of a corresponding set of pixels—that is, dots on the screen.

With the TrueType fonts introduced in System 7, the Font Manager uses equations instead of bitmaps to define the appearance of glyphs. After using the equation to define the outline of a specific glyph in a particular font, the Font Manager translates the outline to a bitmap for display on the screen. The advantage of a TrueType font is that it can be used to generate smoothly rendered glyphs at any size; the instructions included in the font fine-tune the image of the font at different sizes. For example, from one TrueType Courier font, the Font Manager can generate Courier 10-point, Courier 12-point, and Courier 200-point. TrueType fonts are also resolution independent; the same TrueType font can generate glyphs on a 72-dpi device or a 300-dpi device.

Mac OS 8 uses a new font-drawing architecture that's invisible to users and applications. The benefit to application developers is that this architecture—called the **Open Font Architecture** (OFA)—is capable of supporting any type of font format—such as TrueType, PostScript Type 1, and the complex font formats for Asian languages. To bring new fonts to the Mac OS 8 platform, any font developer can provide a **font scaler**—a server program that calculates and renders glyphs at the request of the operating system's imaging services. Because they're implemented as server programs, font scalers are protected from errors in application code. Because operating system services and not applications are the clients for font scalers, an application developer doesn't need to write any additional code to use them.

QuickDraw GX Typography

QuickDraw GX uses the Open Font Architecture, and in addition, provides applications with sophisticated typographic features that extend far beyond the use of multiple font formats. QuickDraw GX allows developers to use typographic shapes that contain multiple fonts and typestyles. The QuickDraw GX line layout facility supports ligatures, kerning, extensive control over line breaks, text alignment and justification, number styles, fractions, subscripts and superscripts, effects such as curved and wavy lines of text, and text containing two or more languages (see Figure 15.3).

A **typestyle** is a variant affecting all glyphs in the same font. Typical typestyles include bold, italic, underline, and so on.

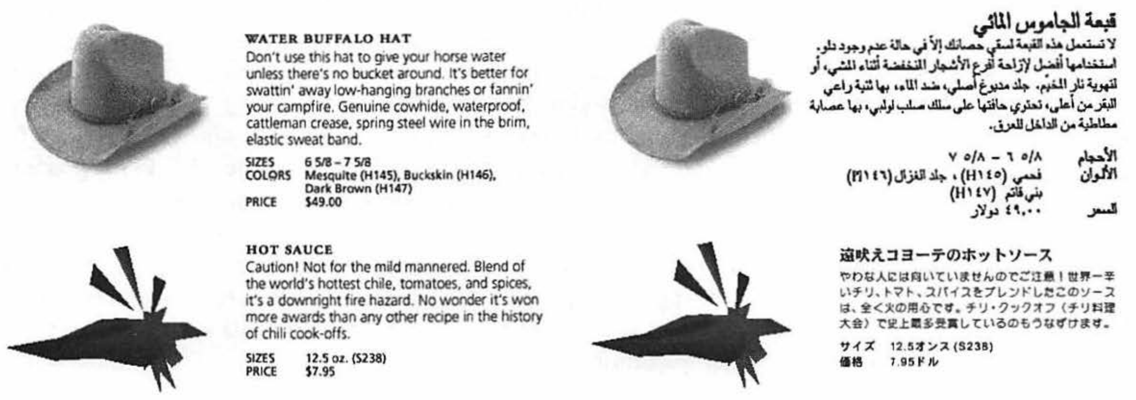
Using QuickDraw GX's layout shapes, a developer can build **ligatures** into a font. (For example, the ligature "*fi*" results from the combination of the letters "f" and "i".) The QuickDraw GX programming interface simplifies the handling of cursor display, text insertion, and spell checking for words containing ligatures. Another traditional typographic feature that QuickDraw GX takes care of automatically is **kerning**—the process of adjusting the spacing between characters so that text has a more aesthetic and natural appearance to the eye.

Using QuickDraw GX, developers can also enhance typestyles by defining variation axes. **Variation axes** are variables whose values consistently change the appearance of a font in terms of weight, width, slant, and the optimal shape for a specific point size.

International Font Handling Support in QuickDraw and QuickDraw GX

The typographic advances of QuickDraw GX don't just serve aesthetics; they're very helpful for internationalizing the text capabilities of Mac OS 8 applications. Chapter 12 discussed how the Human Interface Toolbox helps developers internationalize their human interfaces so that, for example, a

FIGURE 15.3 An internationalized application displaying text in multiple writing systems



WATER BUFFALO HAT

Don't use this hat to give your horse water unless there's no bucket around. It's better for swattin' away low-hanging branches or fannin' your campfire. Genuine cowhide, waterproof, cattleman crease, spring steel wire in the brim, elastic sweat band.

SIZES	6 5/8 – 7 5/8
COLORS	Mesquite (H145), Buckskin (H146), Dark Brown (H147)
PRICE	$49.00

HOT SAUCE

Caution! Not for the mild mannered. Blend of the world's hottest chile, tomatoes, and spices, it's a downright fire hazard. No wonder it's won more awards than any other recipe in the history of chili cook-offs.

SIZES	12.5 oz. (S238)
PRICE	$7.95

قبعة الجاموس المائي

لا تستعمل هذه القبعة لسقي حصانك إلاّ في حالة عدم وجود دلو. استخدامها أفضل لإزاحة أفرع الأشجار المنخفضة أثناء المشي، أو لتهوية نار المخيم. جلد مدبوغ أصلي، ضد الماء، بها ثنية راعي البقر من أعلى، تحتوي حافتها على سلك صلب لولبي، بها عصابة مطاطية من الداخل للعرق.

الأحجام	٦ ٥/٨ – ٧ ٥/٨
الألوان	فحمي (H١٤٥)، جلد الغزال (H١٤٦) بني قاتم (H١٤٧)
السعر	٤٩,٠٠ دولار

遠吠えコヨーテのホットソース

やわな人には向いていませんのでご注意！世界一辛いチリ、トマト、スパイスをブレンドしたこのソースは、全く火の用心です。チリ・クックオフ（チリ料理大会）で史上最多受賞しているのもうなずけます。

サイズ	12.5オンス (S238)
価格	7.95ドル

developer can easily convert an application menu or dialog box from one language to another.

A fully internationalized application also allows a user to enter and edit text in documents using any language. An internationalized program that allows users to create and manage pages on the World Wide Web, for example, could create web pages that display any mix of languages, as shown in Figure 15.3.

However, the text encoding involved in displaying languages within document content can be complicated for a program to handle. For example, fonts for Asian languages may have 8,000 or more glyphs, and several glyphs may be necessary represent a single character. To help developers, QuickDraw GX handles the details of mapping characters to their correct glyphs.

For virtually any writing system in use today, QuickDraw GX handles many text-display, text-editing, cursor-handling, character-highlighting, and character-set concerns, including

- line direction (that is, the direction in which glyphs are read)
- the size of the character set used to represent a writing system
- contextual variation (that is, whether a glyph changes according to its position relative to other glyphs)

QuickDraw GX can also handle right-to-left, left-to-right, and top-to-bottom writing systems, multiple writing systems on the same line, and even writing systems in which adjacent characters sometimes change position.

In Mac OS 8, QuickDraw incorporates support for WorldScript I and WorldScript II. **WorldScript I** supports the display, manipulation, and printing of 1-byte complex text-encoding systems for such languages as Hebrew and

Arabic. (Support for the simpler 1-byte Roman text-encoding system has always been built into the Mac OS.) **WorldScript II** supplies this type of support for 2-byte text-encoding systems, such as Chinese and Japanese. System 7 developers who use WorldScript I or II to create internationalized applications can rely on its availability in Mac OS 8. Whereas text objects (described in Chapter 12) and QuickDraw GX provide the most extensive and the most flexible support for international text in Mac OS 8, many developers are already very familiar with using QuickDraw's text-handling capabilities in conjunction with the multilingual capabilities of WorldScript I and II. Therefore, Mac OS 8 fully supports these technologies for future development as well as backward compatibility.

Anti-Aliased Text Support

In Mac OS 8, both QuickDraw and QuickDraw GX perform anti-aliasing of onscreen text. **Anti-aliasing**, unavailable through the operating system in System 7, is the smoothing of jagged edges on a shape by modifying the shades of individual pixels along the shape's edges. By changing a setting in the Appearance Control panel, Mac OS 8 users turn anti-aliasing on or off. A developer can programmatically turn anti-aliasing on or off within an application; a developer can even turn it on for some portions of a document and off for other portions.

QuickTime Multimedia

Since its inception, the Mac OS has helped users express themselves more effectively by mixing multiple types of media—such as drawings, photographs, and text—into their documents. System 7.5 lets users communicate with documents that incorporate video, music, and interactive, panoramic virtual reality scenes. These System 7.5 features support what's usually referred to as multimedia software—applications that allow users to experience or create information presented in an interactive mixture of video, sound, pictures, and text.

MIDI (Musical Instrument Digital Interface) is a standard protocol for sending audio data and commands to digital devices. **JPEG** (Joint Photographic Experts Group) refers to an international standard for compressing still images. **MPEG** (Motion Picture Experts Group) refers to an international standard for compressing streams of video images.

The underlying architecture for multimedia on the Mac OS is QuickTime. It's a multiplatform standard for viewing, integrating, storing, editing, and playing time-based data, including

- synchronized graphics
- CD-quality sound
- video
- text
- animations
- MIDI sequences

- motion JPEG
- MPEG
- virtual reality scenes
- still images

The QuickTime architecture allows users and developers to work with all of these diverse types of multimedia content as QuickTime movies. A **QuickTime movie** may contain any of these types of time-based data. This flexibility in data handling allows users and developers to focus on creating content rather than on integrating different technologies.

In addition to serving as a playback engine for time-based data, QuickTime includes extensive authoring and editing capabilities, including the ability to manipulate text and capture sound and video. With Apple's MoviePlayer application, for example, users can edit QuickTime movies by cutting and pasting text and video tracks and by editing music clips. Numerous applications from other developers offer highly sophisticated tools for creating and editing QuickTime movies.

QuickTime isn't just for creating and editing video movies. As you'll see in the following sections, other powerful multimedia services are built on top of QuickTime capabilities. Whereas these services are available in System 7.5 as optionally installed system extensions, they're available full-time in Mac OS 8 as shared libraries.

COMPATIBILITY NOTES

The Component Manager

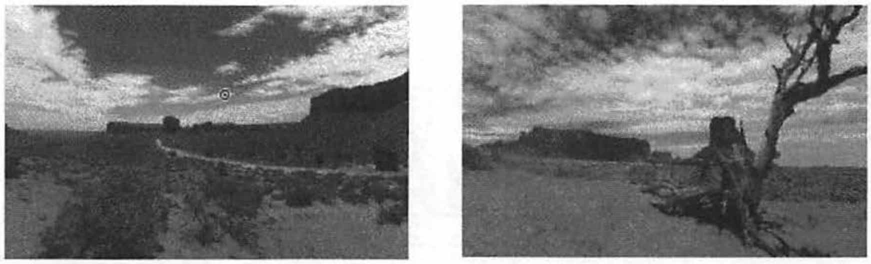
The **Component Manager** is a shared library technology introduced in System 7. In Mac OS 8, this technology is used only by QuickTime as its cross-platform mechanism for loading and unloading libraries of multimedia code. No other portions of the operating system use the Component Manager. Nevertheless, Mac OS 8 supports the Component Manager programming interface as a System 7 application-compatibility service.

QuickTime VR

To let a user move through a panoramic scene or inspect objects from multiple perspectives, a Mac OS 8 application can use QuickTime VR—short for *virtual reality*. The user can zoom in or out of a scene, navigate from one scene to another, and use the cursor to pick up and examine objects. As a user changes the view of a scene, QuickTime VR maintains the correct perspective, creating for the user the effect of being at a location and looking around.

QuickTime VR is built on QuickTime and extends its capabilities. QuickTime VR supplies a panoramic technology that lets users explore 360-degree

Figure 15.4 Two views of the same panoramic scene



scenes and an interaction technology that lets them move and closely inspect individual objects. Figure 15.4 illustrates the panoramic technology. These two views are part of a single 360-degree panoramic scene. The user can pan between these views and look up and down. The CD-ROM version of Figure 15.4 lets you further explore the interactive capabilities of QuickTime VR.

QuickTime VR supports the effects of camera rotation, object rotation, camera movement, and camera zooming. Using the mouse, a user can click on hot spots to interact with a scene, navigate to another location, or activate some action, such as causing an object to rotate.

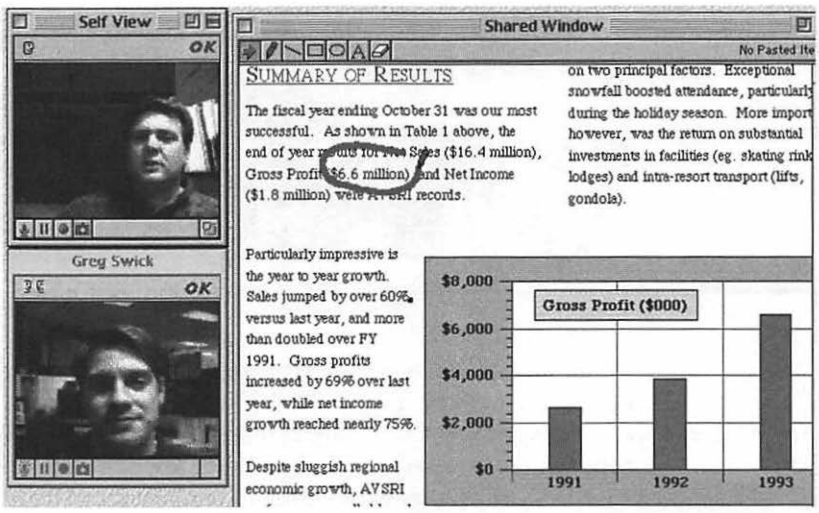
Rather than rendering animations on expensive, high-end workstations, a developer captures a scene with a standard camera and converts it to a QuickTime VR file. Essentially, this is a process of connecting a series of photos in realistic ways. First, a developer captures images with a 35-mm, video, or digital camera and then places the images into the computer in digital form (generally, by using a scanner). The developer then authors, or composes, the panoramic scene, using such techniques as "stitching" (eliminating overlap and smoothing borders between individual images to create a continuous panorama) or "warping" (correcting distortions in perspective from a panoramic camera) and creating interactivity by building in hot spots that link to audio recordings, text, or other panoramic scenes.

Although developers can use a panoramic camera to capture images, one of QuickTime VR's advantages is that it doesn't require such expensive equipment—in fact, 35-mm photos can provide higher resolution and better image depth, and they're easier to digitize. The 35-mm photos also give greater detail than rendered images and allow for more interesting lighting and effects.

Developers can author QuickTime VR scenes once on the Mac OS 8 platform and deliver this content to run on both Macintosh and Windows-based computers. The result is access to a vast market of personal computer users.

The growing use of virtual reality on the Internet is supported by the versatility and realism of QuickTime VR. Because QuickTime VR files are exceptionally small, they're fast to download and don't require much computer disk

FIGURE 15.5 Colleagues collaborating on a document with QuickTime Conferencing



SUMMARY OF RESULTS

The fiscal year ending October 31 was our most successful. As shown in Table 1 above, the end of year results for Net Sales ($16.4 million), Gross Profit ($6.6 million) and Net Income ($1.8 million) were AVSRI records.

Particularly impressive is the year to year growth. Sales jumped by over 60% versus last year, and more than doubled over FY 1991. Gross profits increased by 69% over last year, while net income growth reached nearly 75%.

Despite sluggish regional economic growth, AVSRI

on two principal factors. Exceptional snowfall boosted attendance, particularly during the holiday season. More import however, was the return on substantial investments in facilities (eg. skating rink lodges) and intra-resort transport (lifts, gondola).

space. In fact, a typical panoramic scene can be as small as 200K; thousands of panoramas can fit on a single CD-ROM.

QuickTime Music Architecture

The **QuickTime Music Architecture** helps users work with MIDI music by providing a software synthesizer and a library of musical instruments. Music and synthesizer developers can deliver their own custom software synthesizers and instruments through QuickTime. Mac OS 8 application developers, in turn, can use this architecture to embellish their content with music and create a distinctive aural experience.

QuickTime Conferencing

QuickTime Conferencing is a cross-platform collaboration and communications technology that allows users to broadcast and view real-time digital audio, text, images, and video. Conference participants using an application based on QuickTime Conferencing can share and annotate text, images, screen captures, sound, video, and QuickTime VR scenes. Conference participants can also record their proceedings and transform them into QuickTime movies. Figure 15.5 shows two colleagues with video cameras connected to their computers collaborating over the Internet with QuickTime Conferencing. These two can view live video images of one another and simultaneously work on a shared document.

This conferencing can take place on a variety of networks such as an Integrated Services Digital Network (ISDN), the Internet, local area and wide area networks, and Asynchronous Transfer Mode (ATM) networks. The number of conference participants is limited only by the available network bandwidth.

Although the QuickTime Conferencing technology is Mac OS based, it's designed to fit open standards for interoperability. It's compatible with a wide range of industry-standard video formats and video compression and decompression schemes. QuickTime Conferencing also supports a variety of connection models, including point-to-point video telephony, multiparty video conferencing, broadcast audio and video on an existing LAN, and audio-only or video-only for special applications.

This independence from transport modes, compression schemes, and media devices permits cross-platform video conferencing between Mac OS-compatible computers, other personal computers, UNIX systems, and room-based conferencing systems. Users don't have to worry about whether their hardware equipment, networking equipment, or applications are compatible with the solutions being used on the other end of the conference line.

Mac OS 8 developers can use QuickTime Conferencing as a foundation technology for their video conferencing applications. For example, Apple's own QuickTime Conferencing application, Apple Media Conference, allows users to call other video conference participants over their local area networks (LANS). Those who have a direct connection to the Internet can use these capabilities with remote users as well. The Apple Media Conference application also allows two or more participants to edit the same document. While one participant has the document stored on his or her system, others participants can view it and mark it up in a shared window, as shown in Figure 15.5.

A HISTORY OF MAC OS IMAGING AND MULTIMEDIA FEATURES

The following chronology shows the evolution of imaging and multimedia capabilities inherited by Mac OS 8 from earlier versions of the Macintosh platform.

1984

- The Macintosh computer is introduced, featuring the QuickDraw graphics system and a bitmapped video display.

1985

- Apple introduces the LaserWriter printer.

1988

- ▶ Macintosh computers become the first personal computers to offer plug-and-play support for CD-ROM drives.

1989

- ▶ Color QuickDraw is introduced, making the Macintosh the first personal computer to display photo-realistic images.

1990

- ▶ Macintosh computer models begin including microphones for sound input.

1991

- ▶ With the introduction of System 7, TrueType fonts become available.
- ▶ QuickTime multimedia software is introduced.

1992

- ▶ WorldScript becomes available.
- ▶ Many Macintosh computer models include built-in CD-ROM drives, spurring the growth of multimedia technologies.
- ▶ ColorSync is introduced, making it the first operating system-level implementation of an industry-standard color-matching system.
- ▶ Apple introduces QuickTime for Windows, making QuickTime the first industry standard for developing and using cross-platform multimedia software.

1994

- ▶ The QuickDraw GX imaging system is introduced.
- ▶ QuickTime 2.0 becomes available.

1995

- ▶ QuickTime VR, QuickTime 3D, and QuickTime Conferencing are introduced.

SUMMARY

Mac OS 8 inherits from System 7.5 the imaging technologies of QuickDraw, QuickDraw GX, QuickDraw 3D, and ColorSync. Mac OS 8 also inherits the multimedia technologies of QuickTime, QuickTime VR, QuickTime Conferencing, and the QuickTime Music Architecture. These technologies are fully integrated into Mac OS 8 as memory-efficient shared libraries.

Mac OS 8 adds extra capabilities to the imaging and multimedia technologies inherited from System 7.5. For example, QuickDraw GX is implemented as a reentrant service; QuickDraw and QuickDraw GX perform anti-aliasing of text; QuickDraw incorporates support for WorldScript I and WorldScript II text and font-handling capabilities; and a new architecture for font scalers allows Mac OS 8 applications to display text in potentially any font format. And core operating system features—such as preemptive multitasking, virtual memory, concurrent I/O, faster dynamic storage allocation, and PowerPC optimization—significantly improve the performance of these landmark technologies in Mac OS 8.

PLANNING A PRODUCT FOR MAC OS 8

If you're an application developer, the best way to begin preparing Mac OS 8 products that take advantage of these imaging and multimedia technologies is to adopt these technologies in your System 7 applications. Because the programming interfaces for these technologies remain unchanged in Mac OS 8, any development you undertake with them will run without modification on Mac OS 8.

If you're a developer of printers, you should prepare your products to work in Mac OS 8 by creating QuickDraw GX printer drivers for them.

16

Landmark Networking Technologies

In 1985, the Macintosh computer became the first personal computer to offer built-in networking capabilities. Today, users have come to rely on vastly more powerful Mac OS networking capabilities to collaborate and communicate through such media as shared electronic documents, e-mail, interactive World Wide Web sites, and live video conferencing. Mac OS 8 inherits and improves upon this networking legacy. From System 7.5, Mac OS 8 inherits support for a wide array of networking standards through its Open Transport architecture. Mac OS 8 also inherits popular user-oriented networking facilities from System 7.5, including personal file sharing, QuickTime Conferencing, AppleTalk Remote Access, and Cyberdog.

Personal file sharing allows any Mac OS–compatible computer on a network to be a file server. **AppleTalk Remote Access** allows Mac OS–compatible computers to communicate with network servers over standard telephone lines.

To improve upon its networking legacy, Mac OS 8 offers networking applications the benefits of preemptive multitasking, concurrent network I/O, international language support, and memory protection for server programs. Previous chapters of this book have shown how Mac OS 8 networking applications can take advantage of these core features of the operating system. This chapter describes the specific networking technologies that Mac OS 8 brings forward from System 7.5 and explains how users and developers can take best advantage of them.

If you haven't already done so, read "How to Navigate the Book CD-ROM" in the preface for information about using the CD-ROM version of this book. In the CD-ROM version of the book, the screen shots presented in this chapter are animated.

KEY TERMS AND CONCEPTS

- **Networking** is the sharing of information and services via connected computers. Using communication protocols such as AppleTalk and TCP/IP, networked computers can be linked by various media—for instance, phone lines, LocalTalk cables, Ethernet cables, and radio.
- Network **protocols** are the rules that govern how and in what format data is transmitted between connected computers.
- The **Internet** is a loosely administered worldwide computer network. The Internet is a descendent of the Arpanet, which was created to allow U.S. universities and research institutions to share information easily. The TCP/IP protocols and many of the tools used today on the Internet were first developed for the Arpanet.
- An **intranet** refers to any private network based on Internet protocols and tools. For example, a company might use an intranet to share files and e-mail internally among its employees.
- The **World Wide Web** consists of computers on the Internet that use multimedia to present information and services. These computers also use electronic links within their media to help users quickly find related information and services across the web.
- **Cyberdog** is an OpenDoc-based architecture integrating network services into the Mac OS. Cyberdog allows users to incorporate remotely located information into their software and documents.
- **QuickTime Live!** is a cross-platform venue for live, interactive, online entertainment on personal computers.
- **Open Transport** is the portion of the I/O system that implements industry-standard networking protocols.

MAJOR POINTS OF INTEREST

With the phenomenal growth of the World Wide Web and with users' growing reliance on the Internet, computer networking capabilities are more important today than ever. Mac OS 8 provides powerful networking facilities and, in the tradition of the Mac OS, Mac OS 8 delivers these features in ways that make networking easier than it's ever been—easier for users, easier for application developers, and easier for network content providers.

Mac OS 8 makes networking easier for users in several ways. First, Mac OS 8 provides experts and other types of interactive, online assistance that help users take advantage of the platform's networking capabilities. For example, personal file sharing and network printer sharing were introduced in previous versions of the Mac OS, but a setup expert supplied by Mac OS 8 simplifies the tasks of configuring these features. The setup expert automatically determines important details on behalf of the user, such as whether the user's computer is connected to an AppleTalk network and, if so, what network services are available. The setup expert then asks general questions about the user's goals—for instance, "Would you like to share files over the network?" In response to the user's answers, the setup expert configures these networking features automatically. As described in Chapter 13, any developer can similarly use Mac OS 8 Assistance Services to help users take full advantage of an application's networking features.

AppleTalk refers to a suite of network protocols that have been adopted by many vendors of computers and networking products. AppleTalk networks can be integrated with other network systems, such as the Internet.

Because of the wealth of information that computer networks make available to users, the task of organizing and collecting information has become more complex. As you saw in Chapter 10, the Mac OS 8 Navigation Services make it easier for users to search for any type of information within Mac OS files located on network servers. With Find windows, for example, users can locate and organize network-hosted information relationally, grouping documents with similar content according to user-specified criteria.

With its workspace feature, described in Chapter 12, Mac OS 8 also makes it easier for a user to manage networking preferences, such as e-mail accounts and World Wide Web bookmarks. Several users sharing the same computer can maintain individual networking preferences in their own personal workspaces.

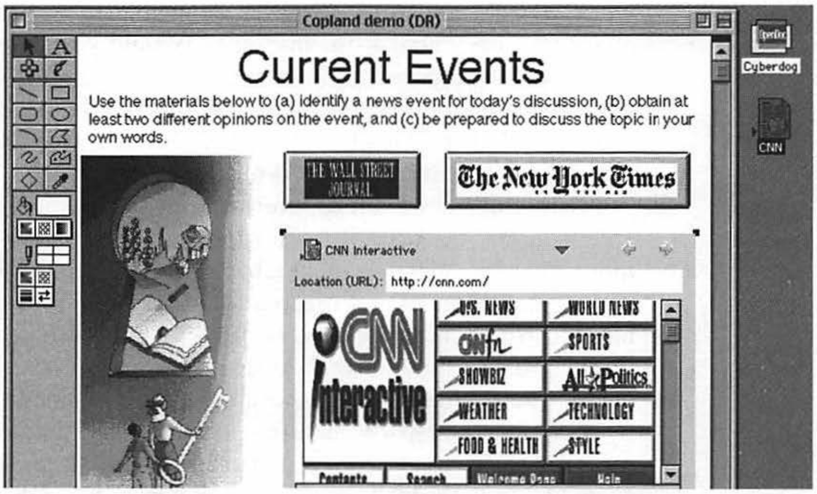
Open Transport makes it easier for developers to incorporate networking services in their applications. By using the programming interface defined by Open Transport, a developer gains access to any type of network to which the user's computer is connected. Open Transport automatically takes care of the communication details appropriate for the user's network.

Cyberdog makes networking easier for users and developers both. For users, Cyberdog provides a consistent, intuitive way to search and browse the Internet and gain access to Internet mail and newsgroups. A user can drop a Cyberdog part into any OpenDoc document to instantly add Internet capabilities to that document. For developers, Cyberdog makes it very easy to provide Internet capabilities within their applications: they need only to support OpenDoc. Then users can extend Internet capabilities to these applications simply by adding Cyberdog parts. Apple supplies users with a set of useful Cyberdog parts, and network application developers can create Cyberdog parts of their own. The Cyberdog classes that developers use to create parts include built-in protocol support and connection support, greatly simplifying the creation of networking products.

As discussed previously in this book, **OpenDoc** is a multiplatform technology for constructing and sharing compound documents, which consist of multiple, user-selected software components, called parts.

Mac OS 8 also makes it easier to write multimedia content for distribution across the Internet. Because QuickDraw 3D and the QuickTime multimedia services described in Chapter 15 are cross-platform, content providers can use these services to create 3D images and various forms of time-based data—such as video, MIDI music, CD-quality sound, and virtual reality scenes—for presentation in web pages that can be shared by Mac OS, Windows, and UNIX users.

FIGURE 16.1 Using a Cyberdog part to open a web page from within a document



Current Events

Use the materials below to (a) identify a news event for today's discussion, (b) obtain at least two different opinions on the event, and (c) be prepared to discuss the topic in your own words.

Location (URL): http://cnn.com/

CYBERDOG

Cyberdog is one of Apple Computer's latest developments. To users, Cyberdog consists of a suite of ready-built OpenDoc parts for accessing and displaying Internet-borne content. To developers, Cyberdog consists of a collection of OpenDoc classes used to create Internet-capable OpenDoc parts.

Cyberdog parts give users a flexible way to manage their access to the Internet. For example, a user can drag and drop icons representing Internet locations into OpenDoc documents. Clicking one of these icons connects the user to an Internet site. The user can also keep a notebook of icons representing regularly visited Internet locations, newsgroups, mail trays, web sites, and other items and services available on the Internet. Clicking these icons connects the user to their associated Internet locations. By dragging and dropping these items into other documents, the user can seamlessly integrate Internet content with other content. And the user can share documents containing this type of content with other users.

Figure 16.1 shows the user opening a Cyberdog part from within a document that contains standard text and graphics. This Cyberdog part contains a browser to a particular web site. The CD-ROM animated version of this figure demonstrates how the user drags an icon representing the web site from the desktop and drops it onto the document. This document becomes a web browser. Every time it's opened, the document creates a network connection and displays the current web page.

Apple Computer supplies users with a suite of useful Cyberdog parts, including

- a World Wide Web browser
- an e-mail reader
- a newsgroup reader
- a Gopher browser
- FTP, the Internet file transfer facility
- Telnet, the Internet remote log-in facility
- utilities for storing and organizing Internet information
- security facilities

Parts are the portions of an OpenDoc document that contain content for view or manipulation by users. At program execution time, **part editors** display part content, facilitate manipulation of the content, and provide a user interface for modifying that content.

With Cyberdog, developers can add Internet connectivity to any existing OpenDoc part (for example, to support e-mail or network file transfers), create new OpenDoc parts with network capabilities, create enhanced replacements for Apple-supplied Cyberdog parts, and even add new network protocols and services to the OpenDoc architecture.

Support for accessing the Internet is built into the Cyberdog classes, simplifying the effort necessary to develop Internet-capable products. For example, a developer can create a Cyberdog part for displaying JPEG data located on the Internet and not worry about network protocols or connections, because Cyberdog handles these details automatically.

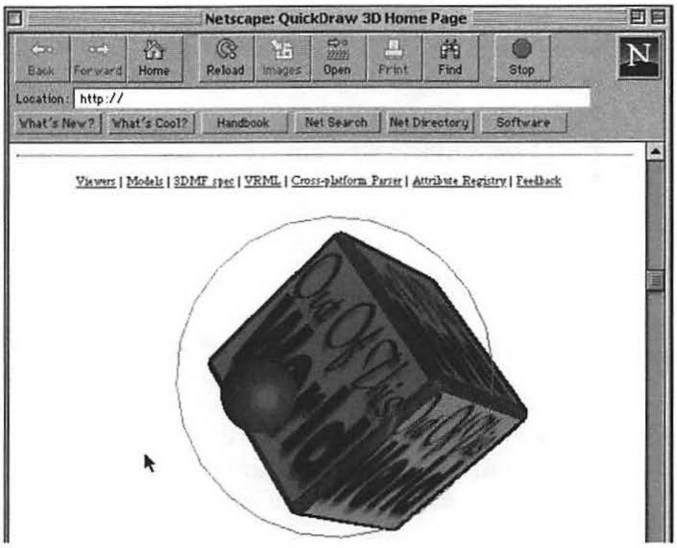
Cyberdog offers all of the extensibility features of OpenDoc, which are described in Chapter 9. From the developer's perspective, a Cyberdog part editor, like an OpenDoc part editor, is a small shared library offering specialized features. Part editors can be created and revised much more quickly than large, stand-alone applications. Cyberdog permits users to dynamically mix and match parts in order to extend networking capabilities according to their own needs and tastes. This extensibility allows developers to quickly create and revise Internet solutions that users can seamlessly integrate into their work environments.

COMPATIBILITY NOTES

PowerTalk and AOCE

PowerTalk and AOCE (Apple Open Collaboration Environment) from System 7 are not supported in Mac OS 8. Instead, these communications and collaboration services are replaced by industry-standard Internet protocols and OpenDoc, the open standard for creating component software like Cyberdog.

FIGURE 16.2 A QuickDraw 3D image manipulated from the Netscape browser


Netscape: QuickDraw 3D Home Page
Back
Forward
Home
Reload
Images
Open
Print
Find
Stop
Location: http://
What's New?
What's Cool?
Handbook
Net Search
Net Directory
Software
Viewers | Models | 3DMF spec | VRML | Cross-platform Parser | Attribute Registry | Feedback

QUICKDRAW 3D AND QUICKTIME MULTIMEDIA ON THE INTERNET

Chapter 16 discussed how Apple Computer's cross-platform 3D and multimedia technologies take advantage of Mac OS 8 capabilities. For content creators, these technologies offer innovative solutions for authoring web pages and publishing information on the Internet. For users, these technologies offer engaging ways to interact with information on the Internet. For example, users can rotate and examine QuickDraw 3D objects displayed on web pages, like the one shown in Figure 16.2.

Central to QuickDraw 3D is the 3DMF (3D Metafile) format. This format allows 3D objects created by one application to be used by other applications, even on different operating systems. This makes 3DMF a convenient mechanism for exchanging 3D graphics across the Internet. To accelerate the implementation of 3DMF across the World Wide Web, Apple Computer licenses 3DMF to developers at no charge for use in their applications. Apple has also proposed 3DMF as a file extension to Moving Worlds VRML 2.0, a new cross-platform standard for dynamic 3D environments on the Internet proposed in March 1996 by Apple, Netscape Communications, and Silicon Graphics.

QuickTime Conferencing, as discussed in Chapter 15, allows users to broadcast and view real-time digital audio, music, text, images, and video on the Internet. Although QuickTime Conferencing technology is Mac OS based, it adheres to open standards for interoperability and permits cross-platform video conferencing between Mac OS, Windows, and UNIX computers and room-based conferencing systems. QuickTime Conferencing users don't have to worry about whether their hardware or applications are compatible with the solutions being used on the other end of a conference line.

As you recall from Chapter 15, QuickTime VR is the virtual reality software that lets a user move through a panoramic scene or inspect an object from multiple perspectives. Content providers use QuickTime VR to create highly interactive web sites. Using this technology, for example, a business can develop a web site that transports consumers into its store and allows them to see a product, zoom in to it, pick it up, turn it around, and navigate to another area of the store.

Integrating all of Apple Computer's interactive multimedia technologies, QuickTime Live! is a showcase of World Wide Web technology. QuickTime Live! presents multimedia webcasts of live entertainment, including images, videos, sound, and QuickTime VR scenes. A QuickTime Live! event is interactive: at custom kiosks, attendees at the event can post pictures of themselves on the Internet and chat with online fans, and online fans are able to see the live entertainment as it's happening. Webcasts can also be archived for people who want to view the event after it takes place.

The February 1996 coverage of the 38th Annual Grammy Awards inaugurated the first QuickTime Live! event to include high-quality, color broadcasts of both live and prerecorded video over the Internet. Creative content for a webcast like the Grammy Awards is produced as follows:

1. Camera crews videotape a story or event.
2. The content is edited with archival footage, titles, and special effects to produce a broadcast-ready show.
3. The show is sent to a Mac OS–compatible computer for digitizing.
4. The digital video stream is webcast over the Internet to geographically disbursed web servers.
5. These servers send the video signal to Mac OS and Windows users who have tuned in using webcast viewer software available from the web site itself.

OPEN TRANSPORT NETWORK ARCHITECTURE

Open Transport is the I/O family that implements the operating system's networking capabilities. Like the rest of the I/O system in Mac OS 8, Open Transport is completely concurrent. Therefore, multiple network transactions are interleaved so that the CPU doesn't waste valuable cycles waiting for any single transaction to be completed. This concurrency, in conjunction with the multitasking capabilities of Mac OS 8, allows Open Transport to transfer network data efficiently and to increase overall system performance.

To make networking easier for users, Open Transport

- Supplies a consistent human interface for configuring AppleTalk, TCP/IP, and other networks.
- Gives users the ability to reconfigure and restart network services without restarting their computers.
- Allows a computer to be connected to multiple networks simultaneously. For example, suppose a user in an office environment connects a LaserWriter printer to his or her computer using LocalTalk. For file access and e-mail, the user's computer may also be connected to the company-wide network via Ethernet. Open Transport maintains both connections without creating a bridge between the networks. (Before Open Transport, a System 7 user would have to open the Network control panel and change the AppleTalk connection back and forth between LocalTalk and Ethernet.)

TCP/IP (Transmission Control Protocol/Internet Protocol) is the major transport protocol and the network layer protocol used in communicating over the Internet.

To make networking easier for application developers, Open Transport defines a programming interface that provides access to any type of network to which the user's computer is connected. The calls that an application makes to Open Transport depend solely on the nature of the communication, not on the transport mechanism. For example, a developer can use one set of functions for any connection-oriented, transactionless protocol, such as ADSP or TCP, and a different set of functions for any connectionless, transactionless protocol, such as UDP or DDP. In other words, after determining the *type* of protocol that is appropriate for an application, a developer can write networking code without worrying about which protocol family will be used. (By comparison, the Cyberdog architecture, as you've seen, shields developers from having to deal with any network protocol or connection details whatsoever.)

Open Transport supports industry standards at both the hardware and the software levels. In particular, Open Transport is based on these key standards: from the X/Open Group, the X/Open Transport Interface (XTI) and the Data Link Provider Interface (DLPI); and from UNIX System V, STREAMS. Open Transport includes implementations of the AppleTalk and TCP/IP protocols and support for common data links, such as LocalTalk, Ethernet, and Token Ring. Open Transport for Mac OS 8 also includes support for serial communications of AppleTalk and TCP/IP network data with an implementation of the Point-to-Point Protocol (PPP).

Compatibility Notes

AppleTalk Manager and MacTCP Programming Interfaces

For compatibility with applications and networking products from System 7, Mac OS 8 supports the AppleTalk Manager and MacTCP programming interfaces as cooperative services. (**MacTCP** allows System 7 applications to communicate on the Internet and on other networks based on TCP/IP protocols.) These programming interfaces can be called only from the main tasks of cooperative programs. Because Open Transport is a reentrant service, and because it's optimized for the PowerPC CPU, developers wishing to create new products based on the AppleTalk or TCP/IP protocols should adopt the Open Transport programming interface.

A History of Mac OS Networking Features

The following chronology shows the evolution of networking capabilities inherited by Mac OS 8 from earlier versions of the Macintosh platform.

1984

- The Macintosh computer is introduced.

1985

- The Macintosh computer becomes the first personal computer with plug-and-play networking using AppleTalk software and LocalTalk cables and connectors.

1989

- The first release of MacTCP provides the foundation for Internet applications on the Mac OS.

1991

- Personal file sharing is built into System 7.
- AppleTalk Remote Access software becomes available, allowing remote computers to connect to AppleTalk networks by telephone.
- Macintosh computers feature plug-and-play Ethernet networking capabilities.

1993

- SNMP, the Internet standard for network management, is available for the Mac OS.

1994

- ▶ System 7.5 incorporates 32-bit TCP/IP Internet support in a new version of MacTCP.

1995

- ▶ Open Transport becomes available for PowerPC-based Mac OS–compatible computers.
- ▶ QuickTime VR, QuickTime 3D, and QuickTime Conferencing operate on the Internet.

1996

- ▶ Open Transport is included with every copy of System 7.5.3.

SUMMARY

Mac OS 8 continues the legacy of Macintosh networking capabilities. Features such as personal file sharing, network printer sharing, and dial-in network access—all of which were introduced in previous versions of the Mac OS—remain integral to Mac OS 8. Workspaces, experts, the Navigation Services, and other facilities introduced with Mac OS 8 make these networking capabilities easier for people to use.

The Cyberdog architecture is one of the more recent—and potentially one of the most revolutionary—networking technologies to be introduced on the Mac OS. By simply dragging and dropping a Cyberdog part onto OpenDoc documents, a user gives them instant Internet capabilities. Cyberdog supplies developers with an extensible architecture for creating small, well-integrated networking products. By handling protocol and network-connection details, Cyberdog simplifies these developers' programming efforts.

QuickDraw 3D, QuickTime, QuickTime VR, QuickTime Conferencing, and other QuickTime technologies help content creators produce cross-platform, interactive forms of multimedia presentations for display and distribution on the World Wide Web.

The operating system provides network I/O support through Open Transport. Developers who want access to network services from within their applications use the programming interface of Open Transport. This programming interface provides a layer of abstraction so that developers don't need to write code specific to any type of networking hardware or transport mechanism.

Concurrent I/O processing, priority-based preemptive multitasking, and memory protection are capabilities newly offered by Mac OS 8 that improve overall performance and system stability for the products using these landmark networking technologies.

PLANNING A PRODUCT FOR MAC OS 8

If you're a developer, you can take the following steps now to take advantage of Mac OS 8 networking capabilities in your software product:

1. Add OpenDoc support to your existing product. In this way, your product becomes Internet capable whenever a user adds a Cyberdog part to one of its documents.
2. Adopt QuickDraw 3D, QuickTime, and QuickTime VR capabilities into your existing application so that content creators can use it for their web authoring and Internet publishing needs.
3. If you're currently developing any new networking applications that require you to deal directly with networking protocols, use Open Transport rather than the AppleTalk Manager or MacTCP programming interfaces.
4. Consider whether your application consumes very much time processing network I/O operations. If so, factor your network I/O code into portions separate from the rest of your application. You can then implement this code more easily as a separate thread of execution in a multithreaded program.
5. Determine whether some portion of your network product would benefit from the extra protection afforded by a separate address space. If so, you should plan to implement this portion as a server program. For example, a program that makes World Wide Web pages available to remote users can be implemented as a server program, thereby preventing programming errors in other applications from disrupting its operations.

Glossary

3DMF (3D Metafile) format A cross-platform format allowing 3D objects created by one application to be used by other applications. The 3DMF format specifies objects, their properties, and properties of the scene that contains them. The file format preserves all object properties in either text or binary formats.

3D object In QuickDraw 3D, a shape that contains or references specifications about how and where the object should be drawn. Developers can create multiple instances of any type of 3D object and assign them individual characteristics. Users can change an object's appearance by rotating it, scaling it, or otherwise transforming it.

3D Viewer A simple mechanism that developers can incorporate into their applications to allow users to view and interact with 3D objects.

access window A window from which the user views and selects help topics described in an Apple Guide guide file.

active window The frontmost modal or document window. Only the contents of the active window are affected by user actions. The active window is identified by distinctive details that aren't visible for inactive windows.

address See **logical address, physical address.**

address space The entire range of memory locations potentially available to a process. Mac OS 8 supports multiple address spaces.

alert box A modal window containing an alert panel used to warn the user or to report an error. An alert box typically consists of text describing the situation and buttons that require the user to acknowledge or rectify the problem. See also **dialog box, modal alert box, movable alert box.**

alert panel A root panel placed inside a modal window to create an alert box. An alert panel contains additional panels, such as for text and buttons.

Alias Manager An operating system service for creating and using data structures for establishing and resolving permanent references to files, folders, and volumes.

alias record A data structure identifying a file, folder, or volume that the user might need to locate again. Alias records assist users and programs in organizing files for easier access. The Alias Manager has algorithms for using alias records to find files that have been moved, renamed, copied, or restored from backup.

A-line instruction See **A-trap.**

AMP See **asymmetric multiprocessor.**

ANSI (American National Standards Institute) An organization devoted to defining commercial standards, such as for programming languages like C.

anti-aliasing The smoothing of jagged edges on a shape by modifying the shades of individual pixels along the shape's edges.

API See **programming interface.**

Appearance control panel A utility allowing the user to choose among the themes available on the system and to modify other aspects of appearance, such as the desktop pattern, highlight color, screen saver, and system font.

Appearance Manager The operating system service that provides the underlying support for themes. The Appearance Manager manages all aspects of themes and theme switching, including the Appearance control panel, support for a variety of color data, and support for animation and sound.

Apple event A data structure used to direct the operation of or communicate information to a program. An Apple event identifies itself and its purpose, names its destination, and contains additional data structures that vary according to the kind of event. Apple events constitute the primary form of interprocess communication in Mac OS 8.

Apple event dispatcher A queue of incoming events and a stack of handler tables containing functions that respond to those events. Mac OS 8 provides a default Apple event dispatcher for every process, and any program may create additional dispatchers as necessary.

Apple event handler A function that extracts pertinent data from an Apple event, performs the action requested by the Apple event, and returns a result.

Apple Event Manager An operating system service that allows programs to send and receive Apple events.

Apple Guide An onscreen help system that explains concepts or guides users through the steps of an operation.

AppleScript A text-based scripting language that lets users control and automate off-the-shelf scriptable programs.

AppleTalk A suite of network protocols that have been adopted by many vendors of computers and networking products. AppleTalk networks can be integrated with other network systems, such as the Internet.

AppleTalk Remote Access A program that allows Mac OS–compatible computers to communicate with network servers over standard telephone lines.

application A program designed to help users accomplish goals; for example, a web browser helps users navigate the Internet, a page-layout program helps users present information in print form, and a flight simulation game offers recreational challenges to users. In Mac OS 8, an application is usually implemented as a cooperative program that may be supported by server programs.

application handler table A table of program-specific Apple event handlers, which a program can add to or remove from its handler table stack at any time. Compare **default handler table.**

application heap A memory area assigned exclusively to a System 7 application for the application's temporary data storage needs. Whereas the Mac OS 8 memory allocators dynamically create additional memory areas to fulfill a program's storage needs, an application heap is fixed in size at application launch time and can't be expanded.

application program See application.

application programming interface See programming interface.

area See memory area.

Assistance Services A group of comprehensive and flexible facilities for delivering help appropriate to users' various goals and skill levels. The Assistance Services include Apple Guide, the Help Manager, the Interview Manager, the Notification Manager, the Tip Manager, and the Trigger Manager.

asymmetric multiprocessor (AMP) Having more than one processor execute instructions so that one processor, the master processor, executes all operating system–related operations. Other processors, called slave processors, perform operations allocated to them by the master processor. Compare symmetric multiprocessor.

asynchronous I/O operation An operation that performs data input or output while the task requesting the operation remains eligible for execution. Compare synchronous I/O operation.

atomic operation A simple routine—such as one that increments or decrements a value, tests and sets a value, or compares and swaps values—that executes to completion; it cannot be interrupted. In Mac OS 8, these operations are implemented using instructions provided by the CPU. Tasks can use atomic operations to help synchronize access to shared data.

A-trap A compiled instruction that is unimplemented by the Motorola 68K family of microprocessors. The first 4 bits of such an instruction have a hexadecimal value of A. For applications compiled to run on 68K-based computers, these instructions invoke routines implemented by the Mac OS.

A-trap table A table containing entry points to Mac OS routines called by code generated to execute on the 68K family of Motorola processors. See also A-trap.

automated assistance Onscreen help that leads a user through an operation and performs as much of the work for the user as possible.

backing provider Code responsible for managing pages of physical memory and transferring data (typically between backing store and physical memory) in response to page faults.

backing store A repository—typically a file on a paging device such as a hard disk—for pages of code or data that aren't currently in physical memory.

backing volume A portion of a storage device used for backing store.

Balloon Help A form of user help that displays, in small windows called help balloons, information about the items to which the cursor points.

bevel button A control that displays information (such as text, icons, or pictures) indicating its purpose. A bevel button can behave like a push button that lets the user press it once to perform an action instantaneously, or it can toggle between selected and unselected states.

bitmap A data structure that represents the positions and states of a corresponding set of pixels.

block copy A mechanism for copying the contents of one memory area to another. See also **interspace block copy.**

blocked task A task that is not eligible for execution until a certain event occurs, such as the completion of a synchronous I/O operation.

block storage device A hardware device that reads or writes blocks of bytes as a group. Disk drives, for instance, can read and write blocks of 512 bytes or more.

bus A path along which information is transmitted electronically within a computer. Buses connect computer devices, such as processors, expansion cards, memory, and peripheral devices.

caption text Text displayed in a static text panel. Caption text can't be changed by the user.

central processing unit (CPU) The microprocessor that executes instructions and transfers information to and from other devices (such as physical memory) over the computer's main bus.

checkbox A control consisting of a small square and its label. The label—which may be text, an icon, a picture, or any other image—indicates what kind of setting the checkbox controls. A checkbox can display off, on, or mixed state settings. The square is checked when the setting associated with the box is in effect, is empty when the setting is not in effect, and contains a short horizontal line when the setting is mixed. A mixed state indicates that a setting is in effect for some elements in a selection and not for others.

class In object-oriented programming, a description of a structure, including both data and methods, used as a template for creating objects.

client library A shared library supplying a client programming interface to an I/O family. A client library typically sends a client request for an I/O service to a family server.

client programming interface A set of routines and data structures defined by an I/O family to allow access to services specific to that particular family. For the file system family, for example, the File Manager supplies a client programming interface allowing program access to files. See also **client library, plug-in programming interface.**

client/server software model A paradigm for designing software that splits computing operations between two entities: clients, which request services, and servers, which provide services. The implementation of client and server entities can take many forms, including processes, tasks, shared libraries, and class objects, which might reside in the same program or in separate programs, on the same computer, or on remote computers connected to a network.

close box A small square on the left side of the title bar of an active window. Clicking it dismisses the window.

CMM See **color management module.**

CMYK color space A color space that models the application of inks and dyes to paper. This color space is based on the colors cyan, magenta, yellow, and black.

coachmark An onscreen graphic that indicates, such as by pointing to or circling, an item on the screen. Coachmarks can be programmatically invoked from Apple Guide guide files as well as from the code for cooperative programs.

code fragment A block of executable code and its static data. Code fragments are created by programming tools at program-generation time. Executable code generated specifically for Mac OS 8 is made up entirely of code fragments. Compare **data-only fragment.**

Code Fragment Manager The operating system service that prepares programs and import libraries for execution.

code section The portion of a code fragment that contains executable code. The Code Fragment Manager maps the code sections of all fragments into read-only areas of system-wide memory. Compare **data section.**

collapse box Appearing in the upper-right corner of a window title bar, an object used for hiding and displaying the contents of the window. The user can click the collapse box once to hide all of the window except the title bar and then click it again to redisplay the entire window.

color management module (CMM) A library that implements color-matching and gamut-checking services.

color space A model, such as RGB and CMYK, for specifying how color information is represented.

ColorSync An industry-standard architecture for moving color images reliably from one device to another (such as from a scanner to a video display and then to a printer) and from one operating system to another.

compiler A tool that converts source code written in a high-level language like C into an object file containing instructions in machine language. See also **linker.**

compile time During program generation, the point at which a software compiler creates object code from source code.

completion routine A routine that's executed as soon as an asynchronous call to some other routine is completed.

Component Manager A shared library technology introduced in System 7. In Mac OS 8, this technology is used only by QuickTime; no other portions of the operating system use the Component Manager. As a System 7–compatibility service, Mac OS 8 fully supports the Component Manager programming interface.

component software See **OpenDoc component.**

composite imaging object An object that combines two or more images of potentially different types.

compound document A single document containing multiple heterogeneous data types, each presented and edited by its own code. A compound document is made up of parts.

concurrent processing The parallel operation of separate pieces of code so that they can share operating system services in a simultaneous or nearly simultaneous manner.

containment hierarchy An arrangement describing which human interface objects are contained by other human interface objects.

context switch The suspension of a currently executing task and resumption of a different task from the point at which it was previously preempted. During a context switch, the microkernel saves the CPU state of the suspended task and restores the CPU state of the task about to resume execution.

contextual menu A pop-up menu containing useful commands and assistance services specific to the item pointed at by the cursor.

control A human interface object that the user can manipulate to take an immediate action or to change a setting to modify a future action.

control panel A System 7 dialog box containing controls that let users specify basic settings and preferences for a systemwide feature, such as the speaker volume. Control panels aren't supported in Mac OS 8.

cooperatively scheduled thread One of multiple paths of execution in a task. Within a task, these threads cooperate by yielding execution control to one another. Cooperatively scheduled threads can be scheduled for execution only when the task containing them is running. Whereas the Mac OS 8 microkernel preemptively schedules all eligible tasks for execution, programs have execution control over the cooperatively scheduled threads they create. A task containing cooperatively scheduled threads may call cooperative services if the task is the main task of a cooperative program. See also **thread.**

cooperative multitasking A policy for sharing the CPU and other system resources among multiple applications. In a cooperative multitasking environment such as System 7, applications cooperate by yielding control of the CPU to one another. Compare **cooperative scheduling, preemptive multitasking.**

cooperative program A program that yields its execution eligibility whenever there are no events pending to which it must respond. When the main tasks of programs cooperate in this manner, the Process Manager synchronizes their access to the Mac OS 8 cooperative services. Usually, a cooperative program is implemented as a standard interactive application or as an OpenDoc document containing various OpenDoc parts, and it presents a human interface. Compare **server program.**

cooperative program address space The address space shared by the processes of all cooperative programs in the system.

cooperative scheduling A policy for scheduling access to the Mac OS 8 cooperative services. When programs cooperate by yielding execution eligibility to one another in their event-handling code, the Process Manager serializes their calls to the cooperative services, thereby allowing each call to execute to completion without being interrupted by another call to the same service. Cooperative scheduling rotates eligibility among the main tasks of cooperative programs so that each can, in turn, be preemptively scheduled with all other tasks in the system. See also **cooperative program, main task.**

cooperative service A shared library supplied by Mac OS 8 for use by programs that cooperate to synchronize access to the library. Programs cooperate by yielding execution eligibility to one another in their event-handling code, allowing the Process Manager to serialize their calls to the cooperative services. Cooperative services support the Mac OS 8 human interface while maintaining compatibility with applications written for System 7. Examples are shared libraries for the Human Interface Toolbox and the QuickDraw graphics system. Compare **reentrant service.**

Cooperative Thread Manager A programming interface used by developers to incorporate cooperatively scheduled threads in their programs.

counting semaphore A synchronization mechanism containing a count variable that may be equal to or greater than zero. Multiple tasks can increment and decrement a counting semaphore. Typically, a task will test whether the counting semaphore is greater than 0 before performing some action involving a shared resource.

CPU See **central processing unit.**

Cyberdog An OpenDoc-based architecture integrating network services into the Mac OS. Cyberdog allows users to incorporate remotely located information into their software and documents.

data fork For a document file, the part of the containing data accessed by programs using a programming interface to the file system, such as the File Manager. This data usually corresponds to data entered by the user; the application creating a file can store and interpret the data in the data fork in whatever way is appropriate. For an application file compiled to run on PowerPC-based computers, the data fork contains the application's code. Compare **resource fork.**

data-only fragment A block of static data created by programming tools at program-generation time. Containing no imported symbols, a data-only fragment is occasionally used to implement a shared library. Compare **code fragment.**

data section The portion of a code fragment or data-only fragment containing static data, including pointers to functions and pointers to global variables, used by code in the code section of this or another fragment. The Code Fragment Manager typically uses a per-process memory allocator for storing the data section, but the developer of a fragment can also direct the Code Fragment Manager to use the system-wide memory allocator instead. Compare **code section.**

data structure An organization of data arranged in a well-defined manner so that the data can be interpreted and manipulated.

default handler table A handler table containing default handlers installed by the operating system. For example, the default handlers for a cooperative program interpret standard events (such as Mouse Down when the user presses the mouse button) and, if necessary, route them to the appropriate panels within a window. Compare **application handler table.**

deferred task System 7 code that runs during an interrupt and can be postponed for later execution. In lieu of using the Deferred Task Manager to create deferred tasks, Mac OS 8 device drivers use the Timing Services to schedule execution time.

definition version The version of a shared library that defines its external programming interface and data format. Client programs are linked to this version at generation time. Compare **implementation version.**

delegated task A user-scheduled operation that is performed automatically when a programmatically determined set of circumstances occurs. For example, a delegated task could dial an online service and check for mail at 8:00 every morning.

delegation The automatic execution of related operations at user-scheduled times.

descriptive assistance Onscreen help, such as a help balloon, that describes application features.

desk accessory A utility similar to a small application that's always available to users from the System 7 Apple menu. Desk accessories aren't supported in Mac OS 8.

desktop animation A shared library that draws to a screen-saving window or to the desktop.

determinate progress indicator See **progress indicator.**

developer An individual or organization that creates software or hardware products for commercial, in-house, or personal use.

device driver Code that directly controls a hardware device, such as a disk drive. In Mac OS 8, device drivers are implemented as I/O plug-ins—dynamically loaded shared libraries that work within I/O families.

Device Manager A client programming interface that supports device drivers through an I/O family that isn't tailored for any particular type of device.

Device Notification Service An operating system service that alerts portions of the I/O system of dynamic changes in device connectivity—as, for instance, when the user removes a disk drive card from a laptop computer and inserts a modem card in its place.

dialog box A window containing a modal dialog panel or modeless dialog panel. Developers use dialog boxes for special or limited purposes, such as soliciting information from the user before the application carries out the user's command. Compare **alert box, standard document window.**

dialog panel A root panel placed inside window to create a dialog box. A dialog panel contains additional panels, such as for editable text and controls.

disclosure triangle A control that governs the display of items in a list, such as an outline containing subtopics. When the arrow of a disclosure triangle points right, only one item is visible beside it. When the arrow points down, both the original item and its subitems are visible in the list. To toggle between the two states, the user clicks the disclosure triangle.

disk cache A portion of physical memory set aside to temporarily store frequently used information that's permanently stored on disk. Because it's faster for the CPU to read information from physical memory than from a disk, disk cache helps programs run faster.

DLL See **shared library.**

document Any piece of work that the user saves as a separate file. The user creates documents using cooperative programs.

document window A window used for displaying and editing document data (such as graphics and text) or for a modeless dialog box. Document windows appear behind floating windows and modal windows in an application's layer. See also **standard document window.**

drag To position the cursor on a visual interface element (such as the title bar or a window), press and hold the mouse button, and move the cursor to a new position. In general, dragging can have different effects, depending on what's under the cursor when the user presses the mouse button. These can include selecting blocks of text, choosing a menu item, selecting a range of objects, shrinking or expanding an object, or moving an icon or other visual element from one place to another.

Drag Manager An operating system service that supports moving visual elements and their associated data from one place to another.

driver See **device driver.**

Driver and Family Matching Service An operating system service that matches hardware-specific software with the I/O devices available on a computer.

Dynamic Storage-Allocation Service A reentrant Mac OS 8 service that defines a programming interface by which code—such as an application or a device driver—manages memory allocations for its data storage needs. While the Dynamic Storage-Allocation Service supplies memory allocators that implement this programming interface, developers can create memory allocators of their own. Compare **Memory Manager.**

dynamic linking The preparation and use of shared libraries at program-execution time.

dynamically linked library (DLL) See **shared library.**

editable text panel A human interface object that lets the user edit the text it displays. Compare **static text panel.**

embedding panel A human interface object that contains other human interface objects. Developers use embedding panels to assemble compound panels from the standard Human Interface Toolbox panels. Compare **root panel.**

encapsulation In object-oriented programming, the packaging of an object's data and the routines that can act on it in order to protect the data from inappropriate changes. This protection is possible because only the object itself can change its data. To gain access to an object's data, a client must call that object's programming interface.

event A user action or system occurrence requiring a response from a program. Events include keystrokes and mouse clicks from the user, requests from other programs (such requests to print files), or any other activities in the system (such as the completion of I/O operations).

event class The attribute of an Apple event that specifies which group of related Apple events the Apple event belongs to. The event class, in conjunction with the event ID, identifies an Apple event and denotes its purpose.

event group A set of bits used as a primitive synchronization mechanism. For example, Mac OS 8 uses event groups to implement locks. Tasks can set each bit individually or in different combinations to synchronize operations and access to shared data.

event ID The attribute of an Apple event that identifies a particular Apple event within a group of related Apple events. The event ID, in conjunction with the event class, identifies an Apple event and denotes its purpose.

event loop A section of code that processes events in a System 7 application. The event loop repetitively requests events from the Event Manager. When the Event Manager returns events, they're dispatched to application-supplied event-handler routines.

Event Manager An operating system service that System 7 applications use to receive information about actions performed by the user, to receive notice of changes in the processing status of the application, and to communicate with other applications.

exception An error—such as a memory access error—or other special condition that is detected by the CPU during code execution. An exception transfers control from the code generating the exception to another piece of code, usually an exception handler.

exception handler A routine that is invoked in response to an exception.

excluded For a memory area, a permission level prohibiting all read and write access to the area. Compare **read-only**, **read/write**.

execution time The general span of time during which programs run on a computer. Compare **generation time, launch time.**

expert A small program that interviews the user to gather information about goals and preferences. The program uses this information to help the user carry out a complex or seldom used operation. Compare **family expert.**

expert assistance Onscreen help that interacts with the user to perform an operation. The expert solicits information only the user can provide and then uses the computer to automate as much of the operation as possible.

experts window A window from which the user views and selects experts.

extensibility The ability of software to be enhanced with new capabilities without breaking those it already supports. Mac OS 8 offers several extensibility mechanisms for developers to enhance the operating system as well as their own products; these mechanisms include the OpenDoc component software architecture, import libraries, plug-ins, SOMobjects for Mac OS, server programs, and the Patch Manager.

extension See **system extension.**

factoring Using Apple event handlers to separate the code that presents an application's human interface to the user from the code that responds to the user's manipulation of the interface. In a fully factored application, any significant user actions generate Apple events that any scripting component based on the Open Scripting Architecture (OSA) can record as statements in a compiled script.

family expert For an I/O family, code that maintains information about the set of family-controllable services or devices and the set of associated I/O plug-ins available on a given computer; this code can be part of the family server or be its own process. Compare **expert.**

family server A privileged server program that receives, processes, and responds to service requests from clients of an I/O family and, usually, calls an I/O plug-in to process these requests.

family services library A shared library allowing an I/O family's plug-ins to call the I/O family for services. Compare **client library.**

floating window A window that appears in front of document windows and behind modal windows in an application's layer. Developers use floating window for tool palettes, catalogs, and other elements with which the user acts on data in document windows.

Folder Manager An operating system service used by programs to determine or define the location of specially used folders. An example is the Fonts folder, where the operating system stores fonts for the user.

font A complete set of glyphs in one typeface and style.

font scaler A server program that calculates and renders glyphs at the request.

forward-compatible memory guidelines A specification for using a subset of System 7 Memory Manager functions to automatically invoke the higher performance, pointer-based memory allocators available with Mac OS 8.

fragment See **code fragment, data-only fragment.**

File Manager The programming interface used by most Mac OS 8 applications to organize, read, and write data stored on volumes. Because the File Manager is reentrant, any task can call its programming interface. Compare **Low-Level File System Services, System 7 File Manager.**

file mapping The association of a disk file with a memory area so that the file's data is paged between physical memory and the file's permanent location on disk. Thus, the disk version of the file (instead of a separate scratch file) serves as backing store for the file's representation in physical memory. See also **memory-mapped file.**

file system The part of the operating system that manages the reading and writing of information located on all storage devices available to the user's computer system. At the most abstract programming level, the file system offers several types of programming interfaces allowing applications to read, write, and otherwise manage information stored on these devices. The file system organizes information hierarchically into volumes, folders, and files. See also **File Manager, Low-Level File System Services, volume format plug-in.**

file system object An item containing information or sets of related information on a storage device. Files, folders, and volumes are examples of file system objects.

filtered handler table An application handler table that causes events for which it contains no handlers to be suspended in an event queue. After the filtered table has been removed from the handler table stack, the Apple Event

Manager passes any suspended events on to the next handler table in the order in which they were originally received. Compare **unfiltered handler table.**

Finder An application that displays the Mac OS 8 desktop, allows users to browse data, and launch applications.

function A named piece of executable code that carries out some action and returns information about the result of that action.

gamut A measure of the range of the lightness, darkness, and density of colors in a given color space.

generation time The time during which executable code is created from source code using such program development tools as a compiler and linker. Compare **execution time.**

generic driver A hardware device driver that makes its service available through the Device Manager client programming interface rather than through a more specific programming interface from another I/O family.

global instantiation See **system-wide instantiation.**

global variable A named storage location for a modifiable value that can be referenced outside the local scope of statements using that variable.

glyph A component of the graphical representation of a particular character.

graphical user interface See **human interface.**

guard page A memory page given excluded permission, so that no tasks can read from or write to that page. Ranges of guard pages at the beginning and end of memory areas help prevent tasks from inadvertently accessing the wrong memory areas. If a programming error causes a task to reference a guard page, the CPU generates an exception before the erring task can adversely affect the wrong memory area.

guide file A file containing onscreen descriptions and instructions used in conjunction with the Apple Guide help system.

Guide Maker A tool available from Apple Computer for building and testing guide files.

handle A variable containing the address of a nonrelocatable pointer, which in turn refers to the address of a relocatable block of data.

handler table A mechanism that matches Apple events with Apple event handlers.

handler table stack An area of memory containing one or more handler tables. A handler table stack always contains a default handler table and, usually, one or more handler tables installed by a program.

hardware interrupt An exception signaled to the CPU by a hardware device, notifying the CPU of a change of condition in the device. A hardware interrupt causes the microkernel to suspend the currently executing task while the CPU executes the hardware interrupt handler that's indicated by the device generating the interrupt.

hardware interrupt handler Code registered with the operating system to service hardware interrupts. I/O plug-ins that respond to hardware devices supply hardware interrupt handlers. See also **main code section, secondary interrupt handler.**

help balloon A small window containing explanatory information about the onscreen item to which the cursor points. Help balloons look like the dialog bubbles in comic strips.

Help Manager A cooperative service that developers use to provide Balloon Help assistance to users.

HFS (hierarchical file system) Apple Computer's standard volume format. HFS organizes files and folders in a hierarchical—that is, tree-like—structure.

hierarchical file system See HFS.

high-level family An I/O family whose family expert registers itself with the operating system to receive information about the availability of devices that can be controlled by the family. Based on this information, the family expert manages the availability of I/O plug-ins for the family. Compare **low-level family.**

human interface The facilities by which a user interacts with programs running on a computer. Because most human interface elements (such as windows, menus, and icons) are visual in the Mac OS, the term **human interface** is generally synonymous with **graphical user interface.** However, user voice input, sounds that alert the user, and other nonvisual elements are part of the human interface as well.

human interface object An execution-time structure that encapsulates one or more human interface elements, such as a window, a dialog box, a control, or a menu. Human interface objects support such object-oriented programming features as inheritance, subclassing, and polymorphism, and such SOM features as language-independence and release-to-release binary compatibility.

Human Interface Toolbox A collection of services that developers use to implement the standard portions of the Mac OS 8 human interface—for instance, windows, controls, and menus. The use of these services allows programs to present a consistent and standard interface to users.

icon A graphic representation of some human interface element, such as a document, disk, folder, or application.

imaging The construction and display of graphical information. Graphical information can consist of shapes, pictures, and text and can be rendered on devices such as screens and printers. All graphical portions of a multimedia document, for example, are processed and displayed in Mac OS 8 through the use of imaging services available as part of the operating system.

imaging object An object, derived from the imaging objects class, that can draw a specific kind of image data, such as text, icons, or QuickDraw pictures.

implementation version The version of an import library providing data and executable code to code fragments using the library. The Code Fragment Manager prepares this version for use by client code at execution time. Compare **definition version.**

imported symbol A name used in a code fragment. The imported symbol references a discrete element of code or data in an import library.

import library A shared library automatically prepared by the Code Fragment Manager for use by a program at launch time. The Code Fragment Manager prepares an import library to resolve imported symbols in program code that were not resolved at link time. Compare **plug-in.**

indeterminate progress indicator See **progress indicator.**

inheritance In object-oriented programming, the transmission of properties and behaviors from one class to another. Compare **subclassing.**

instantiate To create an instance of something, such as a process or an object, at execution time.

internationalization The process of designing and creating applications with various languages and cultures in mind. Building an internationalized application allows a developer to create and maintain a single code base for that application. A localizer usually changes the data or text of the application's human interface for a particular market, but the source code for the application remains unchanged.

Internet A loosely administered worldwide computer network.

interprocess communication The exchange of information among tasks within processes or between tasks in different processes.

interrupt An exception signaled to the CPU to invoke an interrupt handler.

interrupt handler A routine invoked in response to an interrupt. See also **hardware interrupt handler, secondary interrupt handler.**

interrupt latency The interval between the generation of an interrupt and the execution of its interrupt handler. See also **hardware interrupt, hardware interrupt handler.**

instructional assistance Onscreen help, such as a guide file, that explains a concept or the steps necessary to perform an operation.

interspace block copy A mechanism for copying the contents of a memory area in one address space to a memory area in a different address space.

Interview Manager A cooperative service used by a program to obtain information from users so that the program can automate or delegate operations.

interview panel A window containing a question for the user. For example, an interview panel might ask a user to pick the days and times for performing file backups. Compare **presentation panel.**

interview sequence An onscreen interactive dialog between the user and a program. Programs use interview sequences to solicit information for automating or delegating operations on behalf of the user.

intranet Any private network based on Internet protocols and tools. For example, a company might use an intranet to share files and e-mail internally among its employees.

I/O family A collection of software that provides a distinct I/O service to software clients. An I/O family typically consists of a privileged server program, supported by a collection of shared libraries. The file system and the Open Transport networking services are examples of I/O families. See also **client library, family expert, family server, family services library, high-level family, I/O plug-in, low-level family.**

I/O plug-in A plug-in (that is, a dynamically loaded shared library) that provides a particular implementation of the service offered by an I/O family. Within the file system I/O family, for example, a volume-format plug-in implements file system services for a specific volume format. See also **device driver.**

I/O system The portion of the operating system that transfers data to and from peripheral devices, such as hard disks, modems, speakers, keyboards, and pointing devices.

JPEG (Joint Photographic Experts Group) An international standard for compressing still images.

kernel A program that manages all or most of the operating system services necessary to control a computer. In a UNIX-based operating system, for example, the kernel is a program that supervises task and file management, device input and output, and memory allocation. Compare **microkernel.**

kerning The process of adjusting the spacing between characters so that text has a more aesthetic and natural appearance to the eye.

launch time The period during which the Process Manager builds the process for a program that is starting up. Compare **execution time, generation time.**

layer A mechanism for ordering window display across the operating system. Every application has its own layer for windows display. The layer of the active application overlays the window layer of all other applications so that the active application's windows are visible to the user. See also **sublayer.**

library Computer code stored in a file or set of files for use by a variety of software. A library provides building blocks of code for commonly needed operations. In the Mac OS 8 run-time environment, all libraries are implemented as shared libraries based on code fragments; the Mac OS 8 run-time environment doesn't allow the creation of libraries that aren't sharable (or at least potentially sharable) by more than one program.

ligature The combination of more than one letter into a single typographical shape. For example, the ligature "*fi*" results from the combination of the letters "f" and "i".

linker A tool that creates executable files by linking object files with libraries. See also **compiler.**

link time At generation time, the point at which a linker binds object code with imported libraries to create executable code.

list A panel containing a series of items displayed in a rectangle. The user selects the items from a list.

little arrows A control that displays a pair of arrows and typically accompanies a text box containing a numerical value, such as the date or time. Clicking the up arrow increases the value in the text box, and clicking the down arrow decreases it.

locale identifier Information within a text object that encapsulates an International Standards Organization (ISO) language code (which specifies the language in which the text is to be represented) and an ISO region code (which specifies the geographical region for languages that vary by region).

localization The process of preparing a software product for a specific national or regional market.

localizer A developer who adapts applications for particular languages and cultures.

lock A data structure used to synchronize access to a shared resource such as the contents of memory locations. Only the task holding a lock is allowed to modify the data associated with the lock. A **simple lock** prevents other tasks from acquiring the lock until the task holding it has released it. A **read/write lock** allows one or more tasks to acquire the lock for the purpose of simultaneously reading data, but this type of lock allows no more than one task to modify the data at a time. For a lock to protect data, clients of the data must observe the conventions established for the lock. See also **event group.**

logical address A memory address used by code at execution time. The logical address might, in turn, be translated into a physical address by the CPU.

low-level family An I/O family whose family expert has information about a specific piece of hardware, such as a bus or a main logic board. The expert knows how physical devices are connected to the system and can detect when a device that can be controlled by the family is added or removed.

Low-Level File System Services A shared library that manages volume format plug-ins and provides a programming interface for application access to the storage devices connected to the user's system. The Low-Level File System Services define a complex but powerful programming interface from which the File Manager, the System 7 File Manager, and standard C file–I/O routines are abstracted. Developers can directly use the Low-Level File System Services programming interface to build custom facilities for performing file I/O operations.

MacTCP A cooperative service allowing System 7 applications to communicate on the Internet and on other networks based on TCP/IP protocols.

main code section The code within an I/O plug-in that does most of the work of responding to client requests. All I/O plug-ins have a main code section. See also **hardware interrupt handler.**

main task The first task created by the operating system for a process. The main tasks for cooperative programs can safely use Mac OS 8 cooperative services, whereas all other tasks in Mac OS 8 must use only reentrant services.

manager A library or set of related libraries that defines a programming interface to the Mac OS. For example, the Memory Manager is a library of routines that helps developers allocate and release memory for programs running in the System 7 version of the Mac OS.

memory address See **logical address, physical address.**

memory allocation A range of logical addresses used for storing a particular piece of data, such as a global variable or a data structure. A memory allocation can range in size from 1 byte to multiple pages.

memory allocator A plug-in used by client code, such as an application or a device driver, for creating, expanding, and deleting memory areas and for acquiring memory allocations from these areas. As memory areas become fully allocated, a memory allocator automatically creates new memory areas and thereby supplies additional storage to its client code. Mac OS 8 provides three memory allocators: a per-process memory allocator, a system-wide memory allocator, and a nonpageable-memory allocator. For any special needs, Mac OS 8 developers can create their own memory allocators.

memory area A range of logical addresses within an address space. To provide the memory resources required by tasks, Mac OS 8 automatically creates memory areas during program-execution time. Developers can also create memory areas and specify attributes suitable for the needs of their software.

Memory Manager A cooperative service used by System 7 applications to dynamically acquire and release memory allocations. See also **Dynamic Storage-Allocation Service, forward-compatible memory guidelines.**

memory-mapped file A disk file whose contents are mapped into a memory area. The virtual memory system transfers portions of these contents from the file's permanent location on disk to physical memory as needed in response to page faults. Thus, the disk file (instead of a separate scratch file) serves as backing store for the code or data not immediately needed in physical memory.

menu A panel that lets the user view or choose an item from a list of choices or commands. Like all panels, menus are displayed in windows. Unlike other panels, however, developers don't write code to instantiate these windows. Instead, when an application instantiates a menu, the Human Interface Toolbox automatically creates a window for it, displays it in that window, and performs all window management on behalf of the menu.

message A unit of structured data used for communication. In Mac OS 8, for example, tasks in separate processes usually use the Microkernel Messaging Service to communicate information by sending and receiving messages.

message service See **Microkernel Messaging Service.**

method A function defined by a particular class in an object-oriented programming environment.

microkernel A program that manages a small but critical subset of the operating services necessary to control a computer. The Mac OS 8 microkernel, for instance, manages processes, their attendant tasks, and other operating system resources associated with tasks, such as memory, synchronization, timing, and messaging. Other operating system services, such as the I/O system and the Human Interface Toolbox, are implemented separately from the microkernel. Compare **kernel.**

Microkernel Messaging Service An interprocess communication mechanism provided by the Mac OS 8 microkernel for transporting data from one task to another. Typically, these tasks are in different processes. The Microkernel Messaging Service allows bidirectional data transfer so that data may be part of a message, and data may be returned in the reply. It is up to developers to establish their own conventions for interpreting the information exchanged with this service. The Apple events messaging service is built on top of the microkernel messaging service.

microkernel queue A mechanism by which one or more tasks notify another task of some occurrence, for instance, the completion of an asynchronous operation. The task being notified examines the microkernel queue for the notification; this task may block until the notification appears. This communication takes place in one direction only; that is, the tasks writing to a microkernel queue don't receive replies from the task reading the microkernel queue.

MIDI (Musical Instrument Digital Interface) A standard protocol for sending audio data and commands to digital devices.

modal dialog box A modal window containing a dialog panel used to elicit an immediate response from the user. See also **alert box, modeless dialog box, standard document window.**

modal window A window that appears in front of all other kinds of windows in an application's layer. Developers use modal windows for modal dialog boxes and alert boxes, both of which require immediate attention from the user. The user can dismiss a modal window only by clicking its buttons. Compare **document window, floating window.**

modeless dialog box A document window containing a dialog panel. A user can move this type of dialog box, make it inactive and active again, and close it. See also **alert box, modal dialog box, standard document window.**

movable alert box An alert box containing a title bar that allows the user to move it. The user can dismiss a movable alert box only by clicking its buttons. A user can generally switch layers while a movable alert box is active by clicking in another application's window or by choosing another application from the Apple or Application menu. Compare **nonmovable alert box.**

movable modal dialog box A modal window containing a dialog panel used to elicit an immediate response from the user. A movable modal dialog box has a title bar that allows the user to move the dialog box around the screen—for example, to examine the part of the screen that it covers. The user can dismiss the dialog box only by clicking its buttons; however, the user can generally switch layers by clicking in another application's window or by choosing another application from the Apple or Application menu. Compare **nonmovable modal dialog box.**

movie See QuickTime movie.

MPEG (Motion Picture Experts Group) An international standard for compressing streams of video images.

multimedia Combining multiple forms of communication to facilitate the transmission of ideas and information. These forms include text, pictures, video, sounds, music, and other types of data.

multiprocessor computer A single computer having more than one processor to execute instructions. See also **symmetric multiprocessor, asymmetric multiprocessor.**

multitask To manage concurrent execution of more than one task. For example, Mac OS 8 multitasks multiple programs to give them efficient access to computer resources. See also **multithreaded.**

multithreaded Having more than one path of execution. For instance, one thread in a multithreaded program might handle user interactions, another thread might perform calculations, and yet a third might perform I/O. See also **thread.**

Name Registry An operating system service that stores the names, characteristics, and relationships of various software (such as plug-ins and server programs), and information about hardware available on the user's system.

Navigation Services A cooperative service used by applications to present a standard human interface for opening and saving files.

networking The sharing of information and services via connected computers. Using communication protocols such as AppleTalk and TCP/IP, networked computers can be linked by various media—for instance, phone lines, LocalTalk cables, Ethernet cables, and radio.

nonmovable alert box An alert box that can't be moved around on the screen by the user. The user can dismiss a movable alert box only by clicking its buttons.

nonmovable modal dialog box A modal window containing a dialog panel used to elicit an immediate response from the user. The user can't move a dialog box of this sort. Compare **movable modal dialog box.**

nonpageable-memory allocator A plug-in instantiated used by privileged code to manage dynamic storage allocations from system-wide memory areas. This memory allocator keeps all of its memory allocations resident in physical memory for the benefit of privileged code, such as a hardware interrupt handler for a device driver, that can't tolerate page faults. Compare **per-process memory allocator, system-wide memory allocator.**

nonprivileged code Code that is executed while the CPU is in user mode. Nonprivileged code is restricted from using various CPU instructions and hardware addresses and from changing data used by critical portions of the operating system. To protect the stability of the operating system, most code that runs in Mac OS 8 is nonprivileged. Compare **privileged code.**

Notification Manager A reentrant service used to inform the user about the status of a program's operations or to inform the user that a program requires attention. See also **user notification, System Notification Service.**

notifier module A shared library containing information about user notifications for program status. Developers incorporate a notifier-picker panel into interview panels or into their application dialog boxes or windows. See also **Notification Manager.**

object In object-oriented programming, an execution-time structure that contains data and routines that operate on that data. An object is an instance of a class, which can be used to create additional instances that constitute separate objects.

object class See class.

OFA See **Open Font Architecture.**

OpenDoc A multiplatform technology, implemented as a set of shared libraries, that facilitates the construction and sharing of compound documents. See also **part.**

OpenDoc component A software module that functions in the OpenDoc environment. Part editors and part viewers are examples of OpenDoc components.

Open Font Architecture (OFA) The font-drawing architecture used in Mac OS 8. OFA is capable of supporting any type of font format, such as TrueType, PostScript Type 1, and the complex font formats for Asian languages.

Open Transport The portion of the I/O system that implements industry-standard communications and networking protocols.

operating system The software that controls and coordinates computer hardware so that programs installed or controlled by users can run efficiently and conveniently. See also **cooperative service, microkernel, reentrant service.**

page (1) The smallest unit, measured in bytes, of information that the virtual memory system can transfer between physical memory and backing store. (2) To transfer pages between physical memory and backing store.

page fault An exception that causes a page of data or code needed by the CPU to be read from backing store into physical memory.

paging device A secondary storage device, such as a hard disk, used for backing store.

panel Any standard Mac OS 8 human interface object—for instance, a button, scroll bar, or editable text field—that can be placed in a window. Panels are implemented with SOMobjects for Mac OS.

part A portion of an OpenDoc compound document. A part consists of document content, plus—at execution time—a part editor that manipulates that content. The document content is data of a given structure or type, such as text, graphics, or video. To a user, a part is a single set of information displayed and manipulated in one or more frames or windows.

part editor An OpenDoc component that can display and change the data of a part. It is the executable code that provides the behavior for the part. Compare **part viewer.**

part viewer A part editor that can display and print, but not change, the data of a part. Compare **part editor.**

Patch Manager A cooperative service that assists developers in modifying routines supplied in import libraries.

PCI (Peripheral Component Interconnect) An industry-standard data bus available on recent models of Mac OS–compatible PowerPC-based computers.

per-context instantiation See **per-process instantiation.**

per-load instantiation See **private-copy instantiation.**

permissions Authorization to gain access to an entity, such as a file, folder, or memory area, for such purposes as reading, writing, or executing that entity.

per-process Exclusively local to, related to, or identified with a single process. Compare **system-wide.**

per-process instantiation The creation of an instance of a shared library's data section for use by a single process. Compare **per-load instantiation, system-wide instantiation.**

per-process memory allocator A plug-in instantiated by Mac OS 8 for every process to supply every process with dynamic memory allocation. For example, when a Mac OS 8 program requests the operating system to provide storage for data structures related to windows, a per-process memory allocator supplies the necessary memory allocation. Typically, all tasks in a process use the per-process memory allocator instantiated for that process. Compare **non-pageable-memory allocator, system-wide memory allocator.**

personal file sharing A facility built into the Mac OS that allows any Mac OS–compatible computer on a network to be a file server.

physical address A memory address represented by bits on a physical address bus. The physical address may be different from the logical address, in which case the CPU translates the logical address into a physical address.

physical memory Electronic circuitry contained in random-access memory (RAM) chips, used to temporarily hold information at execution time. See also **virtual memory.**

picture See **QuickDraw picture.**

plug-in In Mac OS 8, a shared library dynamically located and prepared for use by another code fragment when the code fragment explicitly calls Code Fragment Manager functions. Compare **import library.**

plug-in programming interface A set of routines and data structures defined by an I/O family to allow communication between a family server and the I/O plug-ins belonging to that family. See also **client programming interface, family services library.**

pointer A variable containing the address of a byte in memory. See also **handle.**

polymorphism In object-oriented programming, the ability to call objects of different classes with the same method. For example, a program might use the same method to draw objects defined by different classes.

Pool Manager A programming interface used by privileged code in System 7 for allocating memory. Compare **Dynamic Storage-Allocation Service.**

pop-up button A control associated with a menu. When the user presses the mouse with the cursor over a pop-up button, additional menu items appear.

pop-up window An onscreen container for storing regularly accessed programs, documents, and folders. When closed, pop-up windows are identified by title bars that appear on the bottom of the screen. When the user clicks a title bar, the pop-up window opens to display its contents. When a user clicks the title bar of an open pop-up window, the window collapses again and moves to the bottom of the screen. Alternatively, when the user drags an icon to a pop-up window title, the window automatically opens. When the user drops the item in the window, it collapses again.

POSIX (Portable Operating System Interface) A set of standard operating-system services defined by the Institute of Electrical and Electronics Engineers (IEEE).

preemptive multitasking The ability of an operating system to allocate access to the CPU and other operating system services among multiple tasks, thereby allowing multiple programs to execute in a simultaneous or nearly simultaneous manner. The microkernel uses a set of well-defined rules to schedule tasks for execution. Following these rules, the microkernel can suspend the execution of a task at any time and resume the execution of another. Compare **cooperative multitasking.**

preemptive scheduling A policy by which the microkernel allocates moment-to-moment access to the CPU among all eligible tasks. The microkernel uses a set of well-defined rules to schedule which task should execute at any time. Following these rules, the microkernel can suspend the execution of a task and resume the execution of another. Preemptive scheduling is necessary for preemptive multitasking. Compare **cooperative scheduling.**

presentation panel An Apple Guide help window that describes a concept or step.

printer driver A plug-in that controls how the contents of a document are spooled, rendered, and sent to a specific output device.

Printing Manager A programming interface allowing System 7 applications to print.

private-copy instantiation The creation of a new instance of a plug-in's data section each time a program calls the Code Fragment Manager to prepare that plug-in for program use. Also called **per-load instantiation.** Compare **per-process instantiation, system-wide instantiation.**

privileged code Code that is executed while the CPU is in supervisor mode. Privileged code can execute CPU instructions that are restricted from nonprivileged code and can access hardware addresses invisible to nonprivileged code. Furthermore, the data used by privileged code can be excluded from nonprivileged code. In Mac OS 8, only the microkernel, portions of device drivers, and certain other portions of the operating system are privileged, thereby protecting the stability of the core operating system from possible programming errors in applications and other types of programs.

privileged server program A server program that runs when the CPU's in supervisor mode (giving the program greater access to computer resources) and that operates on data in a protected system-wide memory area reserved for use by privileged code.

process An instance of a program at execution time. A process is characterized by a set of one or more tasks and the memory and other operating system resources allocated to those tasks. Mac OS 8 uses processes for tracking and reclaiming these resources.

Process Manager A Mac OS 8 service that launches, manages, and terminates processes. On behalf of programs using the cooperative services, the Process Manager also synchronizes use of these services.

processor register A named area of high-speed memory located on the CPU.

profile A means of defining the color characteristics of a given imaging device in a particular state.

program A series of statements instructing a computer to perform various operations. A program is either compiled or interpreted. A compiled program is first created in source code, then transformed by a compiler and linker into executable code. An interpreted program, such as an AppleScript script, is not compiled but instead translated for execution by a separate program called an interpreter. At launch time, the operating system instantiates a process for a program so that it can be executed by the CPU. See also **application, cooperative program, server program, device driver.**

program-execution time See execution time.

program-generation time See generation time.

programming interface The functions and data structures defined by one piece of software, such as an operating system service, for use by client software, such as applications and device drivers. The Mac OS 8 programming interface provides access to such services as window management and file management.

progress indicator A control that shows a lengthy operation is taking place. An **indeterminate progress indicator** communicates that an operation is taking place, but this type of indicator doesn't show how long the operation might continue. A **determinate progress indicator,** by comparison, shows how much of an operation has been completed.

property On a storage device, any piece of information or a set of related information stored by the file system. Properties can be simple data items, such as dates, file types, and icon definitions; or they can be expandable sets of information, such as user-entered data. Each property takes up a certain amount of space allocated on a volume.

protocol A rule or set of rules governing how and in what format data is transmitted between networked computers.

push button A control that displays information (such as text, icons, or pictures) indicating its purpose. When the user clicks a push button, it performs an action instantaneously.

QuickDraw A Mac OS 8 and System 7 service that performs onscreen graphics operations. A precursor to the more sophisticated capabilities of QuickDraw GX, QuickDraw remains a fully supported graphics system in the Mac OS.

QuickDraw 3D A cross-platform, interactive 3D graphics technology.

QuickDraw 3D RAVE (Rendering Acceleration Virtual Engine) An optimized hardware abstraction layer that allows programmers to code directly to 3D graphics accelerator cards for maximum performance.

QuickDraw GX A collection of graphics, typography, and printing services in Mac OS 8 and System 7, providing applications with sophisticated color publishing capabilities.

QuickDraw picture An image described by a sequence of QuickDraw drawing commands that have been saved to a file.

QuickTime A collection of cross-platform operating system services that allow applications to control time-based data, such as video and music.

QuickTime Conferencing A cross-platform collaboration and communications technology that allows users to broadcast and view real-time digital audio, text, images, and video.

QuickTime Live! A cross-platform venue for live, interactive, online entertainment on personal computers.

QuickTime movie A set of time-based data that include sound, video, animation, laboratory results, financial data, or a combination of any of these.

QuickTime Music Architecture A design for creating and playing MIDI music in applications.

QuickTime VR A cross-platform service offering two kinds of virtual reality experiences: a panoramic experience enabling users to explore 360-degree scenes, and an interaction experience allowing users to "pick up" and interact with objects.

radio button A control consisting of a circle with an accompanying label, which may be text, an icon, or a picture. A radio button can draw itself in three different states: on, off, or mixed. (A mixed state indicates that a setting is in effect for some elements in a selection and not for others.) Only one radio button in a group of radio buttons can be on at any one time.

radio button group A panel that encapsulates several radio button panels. Unlike the individual radio buttons, a radio button group panel can handle mouse and keyboard interaction, including highlighting and the tracking of user interactions.

RAM See physical memory.

RAVE See QuickDraw 3D RAVE.

read-only A permission level granting access to view but not change information. Compare excluded, read/write.

read/write A permission level granting access to view and change information. Compare excluded, read-only.

reduced instruction set computing See RISC.

reentrancy The ability of code to process multiple interleaved requests for service nearly simultaneously. For example, a reentrant function can begin responding to one call, become interrupted by other calls, and complete them all with the same results as if the function had received and executed each call serially.

reentrant service A Mac OS 8 operating system facility that can be used concurrently by several pieces of code. For example, the microkernel is a reentrant service that provides concurrent access to the CPU, and the input-and-output (I/O) system is a reentrant service that provides concurrent access to devices like hard disks. Some reentrant services are implemented as shared libraries and others as server programs. Compare **cooperative service.**

resource Any data stored according to a defined structure in a resource fork of a file. The data in a resource is interpreted according to its resource type. This data usually corresponds to data created by the developer for use by the program, but it may also include data created by the user while the application is running.

resource fork The portion of a file that contains the file's resources. Compare **data fork.**

RGB color space A color space based on the red, green, and blue intensities that make up a given color. RGB color spaces are used mainly for displays and scanners.

RISC (reduced instruction set computing) A microprocessor design featuring the rapid execution of simple machine instructions.

root panel An embedding panel that fills a window's content area and to which the window passes all events that affect the window's content. The root panel in turn passes events to other panels that it contains. For example, a modal dialog panel is a root panel that tracks user interaction with the panels it contains and takes care of all event handling required to enforce its modal state.

routine A named piece of executable code that carries out some action. A routine that returns information about the result of that action is called a **function.**

run-time environment The set of conventions that arbitrate how software is generated into executable code, how code is mapped into memory and, at execution time, where data is stored, how data is addressed, and how functions call one another. For Mac OS 8, these conventions are implemented by software development systems (such as the compilers and linkers that generate executable code), the Code Fragment Manager (which manages the preparation of executable code), the microkernel (which schedules code for execution), and the CPU (which executes code).

Scrapbook A program that lets users store text, graphics, sound, movies, 3D objects, and other frequently used information.

SCSI (Small Computer System Interface) An industry standard parallel data bus for connecting computers with peripheral devices.

scratch file Backing store for temporary data not associated with a permanent disk file. A scratch file expands and shrinks dynamically in response to system demands. Compare **memory-mapped file.**

scratch space See swap space.

script A series of statements, written in a scripting language such as AppleScript, instructing a computer to perform various operations. Scripts are translated for execution by interpreter programs.

scriptable In the Mac OS, the ability of programs to be controlled by scripts. The operations of multiple scriptable programs can be coordinated and automated by users of scripting languages such as AppleScript.

scripting language A programming language, such as AppleScript, designed to automate and control programs and to be easier to learn and use than complex programming languages like Pascal or C. The scripts written in scripting languages are translated for execution by interpreter programs.

scroll bar A control that an application embeds in a window to allow a user to change the portion of a document displayed within that window.

scrolling list A list embedded in scrolling panel.

scrolling panel A human interface object containing a vertical scroll bar, horizontal scroll bar, or both. A scrolling panel is designed to contain another panel (such as an editable text panel or the list panel) that's larger than the area allocated for the scrolling panel. Scrolling panels supply functions that allow a developer to set and get vertical and horizontal scroll values, vertical and horizontal scroll increments, and scroll bar visibility.

secondary interrupt handler A routine that runs as a result of an interrupt sent to the microkernel by a hardware interrupt handler or by some other privileged code. A secondary interrupt handler always runs in supervisor mode. A secondary interrupt handler can be preempted only by hardware interrupt handlers, and it always runs as privileged code.

semaphore See **counting semaphore.**

server program In Mac OS 8, a program that has no direct interaction with users and, typically, provides services to other programs along the client/server model. A nonprivileged server program operates on data within its own protected address space, whereas a privileged server program operates on data in a protected system-wide memory area reserved for use by privileged code. Compare **cooperative program.**

shape In QuickDraw GX, an encapsulated data structure capable of defining multiple types of graphic and typographic images—such as lines, points, rectangles, polygons, curves, multiple-curve paths, text, line layouts, and pictures.

shared library A code fragment exporting a set of routines or static data that can be called by multiple programs. Because they are prepared for use dynamically—that is, at program-execution time instead of at program-generation time—shared libraries are also called dynamically linked libraries. See also **definition version, import library, implementation version, plug-in.**

shared memory area A memory area addressable within two or more address spaces. Shared memory is useful for sharing data across a limited number of address spaces. Shared memory can reside at the same address in various address spaces, or it can reside at different addresses. Also, shared memory can have different permissions in different address spaces. Compare **system-wide memory area.**

size box Appearing in the lower-right corner of a window, an object that a user drags to resize the window.

slider A control that displays a range of values, magnitudes, or positions. A movable indicator shows the current setting. Sliders, which can be vertical or horizontal, allow users to alter a value by moving the indicator up and down or back and forth.

Small Computer System Interface See SCSI.

SMP See **symmetric multiprocessor.**

SOM See System Object Model.

SOMobjects for Mac OS The Apple Computer implementation of the System Object Model (SOM), an industry-standard architecture for the development and packaging of object-oriented software. SOMobjects for Mac OS provides the underlying technology for many parts of the operating system. For instance, windows and panels in the human interface are instantiated as objects derived from a class library based on SOMobjects for Mac OS.

stack A memory area where a task stores some of its temporary variables during execution. For code generated to run in Mac OS 8, most routines receive parameters and return results in PowerPC microprocessor registers instead of on a stack. (Compilers automatically handle stack and register conventions when generating executable code.) For example, when a task calls routines, their parameters, local variables, and return addresses may be loaded into a stack. Stacks automatically grow and shrink dynamically as needed.

standard document window A window in which the user enters text, draws graphics, or otherwise enters or manipulates data.

static data Variables and other data for which memory is allocated once so that such data persists between calls to a code fragment.

static image panel A panel that may contain icons, pictures, patterns, and caption text. A static image panel doesn't respond to direct user interaction.

static text panel A panel containing caption text, which can't be changed by the user. Compare **editable text panel.**

subclassing In object-oriented programming, the derivation of a new class from any existing class by adding to or overriding selected data structures and methods defined by the original class. Compare **inheritance.**

sublayer A mechanism for ordering window display for an application. Every layer for an application may contain three sublayers: a sublayer for modal windows, a sublayer for floating windows, and a sublayer for document windows. Each sublayer determines how a window appears in relation to other windows for that application.

supervisor mode A state of operation for the PowerPC processor that allows access to critical processor resources, for instance, all processor instructions and tables that control memory protection. Only the Mac OS 8 microkernel, other portions of the operating system, and portions of device drivers execute while the processor is in supervisor mode. See also **privileged code, user mode.**

swap space In System 7, a single, preallocated area of backing store used for the temporary storage of all data paged out of physical memory. Compare **scratch file, memory-mapped file.**

synchronous I/O operation An operation where a task, after requesting data input or output, is blocked from execution until the data has been fully read in or written out. Compare **asynchronous I/O operation.**

symmetric multiprocessor (SMP) Having several processors in an environment where each processor executes its own tasks and its own copy of the operating system and communicates with the other processors as needed. Compare **asymmetric multiprocessor.**

System 7 The major Mac OS release preceding Mac OS 8.

System 7 File Manager A cooperative service provided for backward compatibility with System 7 applications. Its programming interface can be called only by the main tasks of cooperative programs. Compare **File Manager, Low-Level File System Services.**

system extension A file in System 7 containing code that's loaded into memory at system startup time. System extensions aren't supported in Mac OS 8.

system heap A memory area in the cooperative program address space reserved for various data structures used by the Process Manager and other portions of the operating system in support of System 7 applications.

System Notification Service A set of reentrant services that allows one task to broadcast information about a change in the state of the system. Any number of other tasks can subscribe to its notifications. For example, the device driver for a display screen can use the System Notification Service to announce that the user has changed screen resolutions or bit depth. Code relying on the resolution or color capabilities of the device can then take action based on the notification.

System Object Model (SOM) An industry-standard architecture licensed by IBM for the development and packaging of object-oriented software. See also SOMobjects for Mac OS.

system-wide Pertaining to all processes in all address spaces. Compare **per-process.**

system-wide instantiation The creation of an instance of a shared library's data section in a system-wide memory area so that all tasks in the system can potentially use the same instance of the data. Compare **per-process instantiation, private-copy instantiation.**

system-wide memory allocator A plug-in that can be used by any type of code to acquire memory allocations from system-wide memory areas. Compare **per-process memory allocator, nonpageable-memory allocator.**

system-wide memory area A range of addresses that appear at the same location of every address space, making the contents of that area visible in all address spaces. Compare **shared memory area.**

task (1) The basic unit of program execution in Mac OS 8. Preemptively scheduled and assigned a priority by the microkernel, every task has its own stack and set of registers. The microkernel uses processes to track the resources required by tasks so that every process is associated with at least one task and several tasks can be associated with a single process. See also **main task.** (2) A sequence of actions that can be triggered programmatically on behalf of the user. Usually called a **delegated task.**

TCP/IP (Transmission Control Protocol/Internet Protocol) The major transport protocol and the network layer protocol used in communicating over the Internet.

text-encoding specification Information within a text object that identifies the text-encoding system used for text within the object.

text-encoding system A computer representation for one or more character sets used by one or more languages and regions. For instance, Unicode is a 16-bit text-encoding system that provides a code for every character in every major writing system.

text engine A shared library that manages the formatting, drawing, and editing of the text in response to user actions and application calls to an editable text panel.

text object A system-wide data type used as the fundamental unit of text interchange in Mac OS 8. A text object consists of text, a text-encoding specification, and a locale identifier. Text objects allow developers to manipulate text without dealing with the details of various text-encoding systems.

theme A coordinated set of human interface designs that determine the appearance of human interface objects on a system-wide basis.

thread (1) A path of execution. For example, one thread in a program might handle user interactions, another might perform calculations, and a third might perform I/O. (2) To design software with more than one path of execution. Mac OS 8 developers can thread products using one or a combination of three different approaches. That is, developers can divide operations so that they are performed by more than one process, by more than one task in a single process, or by more than one cooperatively scheduled thread within a single task.

Thread Manager See Cooperative Thread Manager.

three-dimensional object See 3D object.

Time Manager task System 7 code scheduled for execution independent of CPU clock speed or the occurrence of hardware interrupts. In lieu of using Time Manager tasks, Mac OS 8 device drivers use the Timing Services to schedule execution time.

time slice An interval of time during which a task is given access to the CPU. In general, Mac OS 8 uses priority-based scheduling. However, when multiple tasks have the same priority and that becomes the highest priority on a system, Mac OS 8 allows each task to execute for its time slice. When a time slice expires, the microkernel switches to the next task with the same priority.

Timing Services Operating system facilities used to schedule the execution of device driver code at particular times.

tip A suggestion for making more efficient use of application features. Users decide whether tips are displayed and how the Tip Manager signals their presence. For a user who elects to see tips, the Tip Manager prepares them for display whenever Mac OS 8 detects that the tips could benefit the user.

Tip Manager A Mac OS 8 service for providing users with suggestions about making more efficient use of program features.

title bar icon A small icon used for drag-and-drop operations involving the document displayed in a window. For example, a user can drag a document's title bar icon to a folder on the desktop, then drop it to save the document in that location.

Toolbox See Human Interface Toolbox.

Transmission Control Protocol/Internet Protocol See TCP/IP.

trigger condition The event or state—such as a specific time, time interval, or other programmatically determined condition—that invokes a delegated task. See also **trigger module.**

Trigger Manager A Mac OS 8 service that tracks trigger conditions and invokes delegated tasks.

trigger module A shared library containing information about trigger conditions (that is, events and states necessary to invoke delegated tasks). A program can register a user-selected trigger condition with a trigger module. Whenever circumstances match those specified in a trigger condition, the trigger module sends an Apple event to the registered program, which then performs a delegated task.

typestyle A variant affecting all glyphs in the same font. Typical typestyles include bold, italic, underline, and so on.

typography The arrangement and appearance of printed characters.

unfiltered handler table An application handler table that allows events for which it contains no handlers to be passed to the next handler table in the stack. Compare **filtered handler table.**

user mode A state of operation for the PowerPC processor that protects certain processor resources, such as various processor registers, from being modified. To protect the stability of the user's system, most code in Mac OS 8 runs while the processor is in user mode. See also **nonprivileged code, supervisor mode.**

user notification An audible or visible indication that a program requires the user's attention, or a communication informing the user about the status of a program's operations. User notifications can take such forms as sounds, icons that blink at the top of the screen, reports in a log file, and onscreen alert boxes containing short messages. For example, an e-mail server might display a small dialog box informing the user that new mail has arrived or that no connection was made to an online mail service. See also **Notification Manager, System Notification Service.**

variable A named storage location for a modifiable value.

variation axes Variables whose values consistently change the appearance of a font in terms of weight, width, slant, and the optimal shape for a specific point size.

vertical blanking task (VBL) System 7 code that can be executed during the time a display screen is refreshed.

Vertical Retrace Manager The part of System 7 that schedules and executes code during a vertical retrace interrupt (VBL). In Mac OS 8, vertical blanking interrupt facilities are provided by the video display expert that's responsible for the display family. Code outside the display family may not make VBL calls.

virtual memory Addressable memory beyond the limits of available physical memory. Mac OS 8 extends physical memory by storing on a secondary storage device, such as a hard disk, code and data not immediately required by the CPU.

visual separator A panel that displays horizontal, vertical, or rectangular elements that are used to visually separate other panels in a window. A rectangular visual separator can optionally include a title.

volume A portion of a storage device formatted to contain folders and files. A hard disk, for example, may be divided into several volumes. See also **volume format, volume format plug-in.**

volume format The structure of file and folder information on a disk. Mac OS 8 supports several volume formats, including the hierarchical file system (HFS) and other industry-standard formats, such as the file allocation table (FAT) file system used by DOS and Windows.

volume format plug-in A shared library that organizes information on a storage device. To support different storage devices, several volume format plug-ins may be in use on the user's system. The file system dispatches and routes information between volume format plug-ins and programs that manipulate files and folders.

window A human interface object that presents information such as a document or a message. Windows are implemented with SOMobjects for Mac OS. See also **panel.**

window group A collection of related windows. Whenever the user activates a window that has an associated window group, all the windows in the group also come as far forward as they can while maintaining their current ordering.

workspace One of several separate custom user environments for a single computer. A workspace is characterized by such user-specified attributes as theme, level of complexity, and application preferences.

WorldScript A Mac OS programming model for developing international applications. Encompassing technologies that became available in System 7.1, WorldScript defines an approach to programming and software design that includes the use of human interface design strategies and specific programming interfaces supplied by the operating system.

WorldScript I An operating system service in Mac OS 8 and System 7 that supports the display, manipulation, and printing of 1-byte complex text-encoding systems for such languages as Hebrew and Arabic.

WorldScript II An operating system service in Mac OS 8 and System 7 that supports the display, manipulation, and printing of 2-byte text-encoding systems, such as Chinese and Japanese.

World Wide Web A growing group of computers on the Internet that use multimedia to present information and services. These computers also use electronic links within their media to help users quickly find related information and services across the web.

zoom box An object in a window title bar used for sizing the window. When the user clicks the zoom box once, the window automatically expands to its optimal size on whichever screen is displaying most of the window. Clicking the zoom box a second time restores the window to its previous size and location.

Index

Numbers followed by the letter f indicate figures; numbers followed by the letter t indicate tables.

Macintosh

MAC OS 8 REVEALED

Apple PRESS

With the next release of the Macintosh operating system, Apple will provide a state-of-the-art platform for developers to create new software products. At the core of Mac OS 8 is a redesigned operating system foundation based on microkernel technology that will dramatically increase user productivity.

Author Tony Francis worked closely with the Mac OS 8 engineering team to provide the inside story of the design and development of this innovative technology. The book describes the technical geography of Mac OS 8 and illustrates how the system's user benefits can be implemented in software and hardware products.

Written for developers, system administrators, and information systems professionals, ***Mac OS 8 Revealed*** explains the key technologies of Mac OS 8:

- How Mac OS 8 exploits preemptive multitasking to perform many operations concurrently;
- How the new memory protection model insulates the microkernel and other operating system services to provide system stability;
- How to provide automatic, intelligent assistance to users;
- How to allow the user to customize and scale the human interface.

The accompanying CD-ROM contains an electronic version of the book with animated illustrations that demonstrate the capabilities of Mac OS 8. Simply click on a screen shot marked with a film strip in the book and see how a new feature in the operating system works.

System requirements: Macintosh computer with 68040 processor or faster; CD-ROM drive; System 7.5 or later; 2 MB of free RAM (5 MB required to view the electronic version of the book).

Tony Francis has been with Apple for over 10 years as a lead writer on the *Inside Macintosh* team.

Cover design by Suzanne Heiser

53495

9 780201 479553

ISBN 0-201-47955-9

$34.95 US
$48.00 CANADA

Addison-Wesley Publishing Company, Inc.
Find A-W Developers Press on the World Wide Web at:
http://www.aw.com/devpress/